3I1 – 2007/2008

HC12

Manuel Technique

SOMMAIRE

CPU12

Registres du CPU **5**
Liste des instructions du HCS12 **7**
Résumé des principales instructions **9**
Vecteurs d'interruption **19**

Moniteur D-Bug12

Principales commandes de D-Bug12 **22**
Routines utilisateur **31**
Relais RAM des vecteurs d'interruption **37**
Carte mémoire de D-Bug12 en mode EVB **38**
Adresses d'interruption **39**

Registres du microcontrôleur

Registres du cœur **41**
Registres du Timer **44**
Registres du convertisseur A / N ATD0 **48**
Registres de gestion des PWM **50**
Registres interface série asynchrone (SCI0 SCI1) **50**
Registres interface série synchrone (SPI0) **52**
Registres des ports d'Entrée Sortie logiques (PIM) **53**

Carte d'application EVB12

Implantation des composants **57**

Annexe

Codage des instructions du HC12 et de l'adressage indexé **61**

CPU12

Les informations qui suivent sont tirées du fichier :

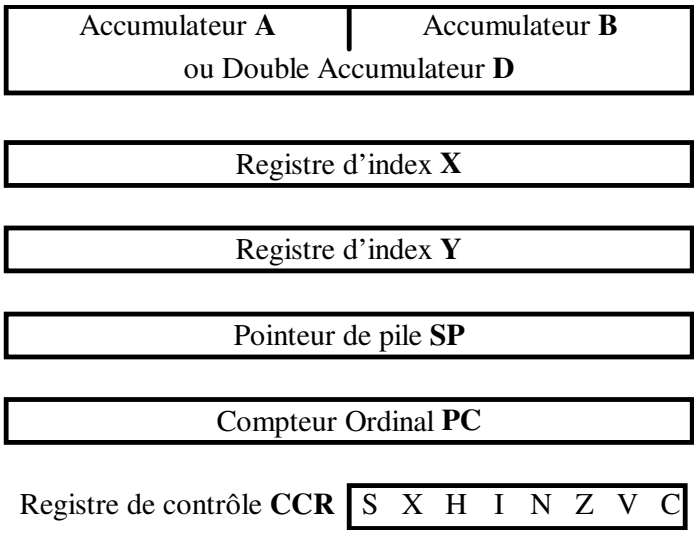
CPU12RG.PDF

qui fournit sous forme condensée la liste et le mode d'emploi des instructions de programmation du CPU. D'autre part, le fichier :

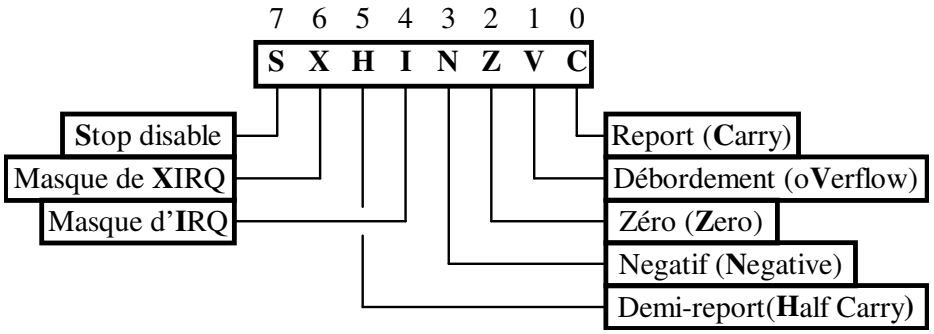
CPU12RM.PDF

constitue le manuel de référence auquel on se reportera pour tout renseignement complémentaire sur le fonctionnement du CPU.

Registres du microprocesseur HCS12



Registre de contrôle CCR



Codes ASCII

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

LISTE DES INSTRUCTIONS DU HCS12

Opérations d'affectation

- Générales :	**CLR**,**CLRA**,**CLRB**	Mise à 0
	LDAA,**LDAB**,**LDD**,**LDS**,**LDX**,**LDY**,**LEAS**,**LEAX**,**LEAY**	Chargement
	STAA,**STAB**,**STD**,**STS**,**STX**,**STY**	Rangement
	MOVB,**MOVW**	Transferts mémoire
	SEX,TAB,TBA,TAP,**TFR**,TPA,TSX,TSY,TXS,TYS	Transferts de registres
	EXG,XGDX,XGDY	Echanges de registres
- De bits :	CLC,CLI,CLV,SEC,SEI,SEV	Choix d'indicateurs
- De pile :	**PSHA**,**PSHB**,**PSHC**,**PSHD**,**PSHX**,**PSHY**	Mise en pile
	PULA,**PULB**,**PULC**,**PULD**,**PULX**,**PULY**	Retrait de pile

Opérations arithmétiques et logiques

- Arithm. :	**ADCA**,**ADCB**,**ADDA**,**ADDB**,**ADDD**	Addition
	SBCA,**SBCB**,**SUBA**,**SUBB**,**SUBD**	Soustraction
	MUL,**EMUL**,**EMULS**,**EMACS**	Multiplication
	EDIV,**EDIVS**,**FDIV**,**IDIV**,**IDIVS**	Division
	DEC,**DECA**,**DECB**,DES,**DEX**,**DEY**	Décrémentation
	INC,**INCA**,**INCB**,INS,**INX**,**INY**	Incrémentation
	CMPA,**CMPB**,**CPD**,**CPS**,**CPX**,**CPY**	Comparaison
	MAXA,**MAXM**,**MINA**,**MINM**, **EMAXD**,**EMAXM**,**EMIND**,**EMINM**	Minimax
	TST,**TSTA**,**TSTB**	Test (comp à 0)
	ABA,ABX,ABY,**CBA**,**SBA**	Diverses
- Décalages :	**ASL**,**ASLA**,**ASLB**,**ASLD**	Arithmétique à gauche
	ASR,**ASRA**,**ASRB**,**ASRD**	Arithmétique à droite
	LSL,**LSLA**,**LSLB**,**LSLD**	Logique à gauche
	LSR,**LSRA**,**LSRB**,**LSRD**	Logique à droite
	ROL,**ROLA**,**ROLB**	Rotation à gauche
	ROR,**RORA**,**RORB**	Rotation à droite
- Logiques	**ANDA**,**ANDB**,**ANDCC**,**ORAA**,**ORAB**,**ORCC**,**EORA**,**EORB**	Et,Ou,Ouexclusif
	BITA,**BITB**	Test de bits
	NEG,**NEGA**,**NEGB**	Négation
	COM,**COMA**,**COMB**	Complémentation

Ruptures de séquence

- Inconditionnelles :	**BRA**,**BRN**,**JMP**,**LBRA**,**LBRN**	
- Conditionnelles :	**BCS**(**LBLO**),**BCC**(**LBHS**),**LBCS**(**BLO**),**LBCC**(**BHS**)	Déclenchés par C
	BVS,**BVC**,**LBVS**,**LBVC**	Déclenchés par V
	BEQ,**BNE**,**LBEQ**,**LBNE**	Déclenchés par Z
	BMI,**BPL**,**LBMI**,**LBPL**	Déclenchés par N
	BLT,**BGE**,**LBLT**,**LBGE**	Déclenchés par N⊕V
	BLE,**BGT**,**LBLE**,**LBGT**	Déclenchés par Z+(N⊕V)
	BLS, **BHI**,**LBLS**,**LBHI**	Déclenchés par C+Z
- Appel de routine et retour :	**BSR**,**CALL**,**JSR**,**RTC**,**RTI**,**RTS**,**SWI**	

Opérations diverses

- Intégrées sur bits :	**BSET**,**BCLR**	Positionnement multiple de bits
	BRCLR,**BRSET**	Test de bits et branchement si Z
- Intégrées diverses :	**DBEQ**,**DBNE**,**IBEQ**,**IBNE**,**TBEQ**,**TBNE**	Boucles
	ETBL,**TBL**	Interpolation
	MEM,**REV**,**REVW**,**WAV**	Logique floue
- Autres :	**BGND**,**DAA**,**NOP**,**TRAP**,**STOP**,**WAI**	

Les mnémoniques qui ne sont pas en caractère gras sont issus du HC11. Pour des raisons de compatibilité, les assembleurs HCS12 les reconnaissent et les traduisent de la façon suivante :

```
ABX    ⇒    LEAX   B,X      ;X = X+B
ABY    ⇒    LEAY   B,Y      ;Y = Y+B
CLC    ⇒    ANDCC#$FE      ;Carry à 0
CLI    ⇒    ANDCC#$EF      ;Masque d'IRQ à 0
CLV    ⇒    ANDCC#$FD      ;Overflow à 0
DES    ⇒    LEAS   -1,SP   ;SP = SP - 1
INS    ⇒    LEAS   1,SP    ; SP = SP + 1
SEC    ⇒    ORCC   #$01    ;Carry à 1
SEI    ⇒    ORCC   #$10    ;Masque d'IRQ à 1
SEV    ⇒    ORCC   #$02    ;Overflow à 1
SEX    ⇒    voir notice              ;Extension de signe
TAP    ⇒    TFR    A,CCR   ;CCR = A
TPA    ⇒    TFR    CCR,A   ;A = CCR
TSX    ⇒    TFR    SP,X    ;X = SP
TSY    ⇒    TFR    SP,Y    ;Y = SP
TXS    ⇒    TFR    X,SP    ;SP = X
TYS    ⇒    TFR    Y,SP    ;SP = Y
XGDX   ⇒    EXG    D,X     ;Echange de D et X
XGDY   ⇒    EXG    D,Y     ;Echange de D et Y
```

RESUME DES PRINCIPALES INSTRUCTIONS

Ce résumé présente en quelques lignes les principales instructions. Il en existe d'autres qu'on trouvera dans la liste complète du fichier *CPU12RG.PDF.*
On trouvera le mode d'emploi détaillé de chaque instruction dans le fichier *CPU12RM.PDF*.

Notations utilisées

Registres du CPU

Accumulateur A — A ou a	Registre d'index Y — Y ou y
Accumulateur B — B ou b	Pointeur de pile — SP, sp, ou s
Accumulateur D — D ou d	Compteur ordinal —PC, pc, ou p
Registre d'index X — X ou x	Registre de codes de condition — CCR ou c

Expressions italiques de la première colonne (Source)

abc — A ou B ou CCR
abcdxys — A ou B ou CCR ou D ou X ou Y ou SP. Quelques assembleurs autorisent T2 ou T3.
abd — A ou B ou D
abdxys — A ou B ou D ou X ou Y ou SP
dxys — D ou X ou Y ou SP
msk8 — masque 8-bits. Quelques assembleurs exigent le symbole # avant la valeur
opr8i — valeur immédiate 8-bits
opr16i — valeur immédiate 16-bits
opr8a — adresse 8 bits utilisée avec le mode d'adressage direct
opr16a — adresse 16 bits
oprx0_xysp — Post octet d'adressage indexé :

*oprx*3,–*xys*	Prédécrément de X, Y ou SP de 1 à 8
*oprx*3,+*xys*	Préincrément de X, Y ou SP de 1 à 8
*oprx*3,*xys*–	Postdécrément de X, Y ou SPde 1 à 8
*oprx*3,x,y,s+	Postincrément de X, Y ou SP de 1 à 8
oprx5,xysp	Décalage constant de 5 bits par rapport à X, Y, SP ou PC
*ab*d,*xysp*	Décalage accumulateur A, B ou D par rapport à X, Y, SP ou PC

oprx3 — Toute valeur entière positive entre 1 et 8 pour un pré/post incrément/décrément
oprx5 — Tout entier entre –16 et +15
oprx9 — Tout entier entre –256 et +255
oprx16 — Tout entier entre –32,768 et 65,535
page — valeur 8 bits de PPAGE. Certains assembleurs exigent le symbole # avant la valeur
rel8 — Adresse relative de destination d'un branchement entre –256 et +255
rel9 — Adresse relative de destination d'un branchement entre –512 et +511
rel16 — Adresse relative sur 64 K d'espace mémoire
trapnum — Tout entier de 8 bits entre $30 et $39 ou entre $40 et $FF
xys — X ou Y ou SP
xysp — X ou Y ou SP ou PC

Opérateurs

\+ — Addition
\- — Soustraction
• — Et logique
+ — OU logique (inclusif)
⊕ — OU logique (exclusif)
×— Multiplication
÷ — Division
M — Négation. Complément à 1 (inverse chaque bit de M)
: — Concaténation
Exemple: A : B signifie la valeur 16-bits formée en concaténant l'accumulateur 8-bits A avec l'accumulateur 8-bits B. A est le poids fort.
⇒ — Transfert
Exemple: (A) ⇒ M signifie que le contenu de l'accumulateur A est transféré dans l'emplacement mémoire M.
⇔ — Echange
Exemple: D ⇔ X signifie l'échange du contenu de D avec celui de X.

Durée

La colonne durée indique la durée de chaque instruction en cycles machine. Les instructions de branchement conditionnel ont deux durées suivant que la condition est réalisée ou non.*

Notation des modes d'adressage

INH — Inhérent; pas d'opérande dans le code objet
IMM — Immédiat; opérande immédiat dans le code objet
DIR — Direct; l'opérande est l'octet poids faible d'une adresse entre $0000 et $00FF
EXT — L'opérande est une adresse 16 bits
REL — Déplacement relatif en complément à deux pour les instructions de branchement
IDX — Indexé (sans octet d'extension); comprend :
Déplacement constant 5-bits à partir de X, Y, SP, ou PC
Pré/post incrément/décrément de 1 à 8
Déplacement sur l'accumulateur A, B, ou D
IDX1 — déplacement signé sur 9-bits à partir de X, Y, SP, ou PC; 1 octet d'extension
IDX2 — déplacement signé sur 16-bits à partir de X, Y, SP, ou PC; 2 octets d'extension
[IDX2] — Indirect indexé; déplacement 16-bits à partir de X, Y, SP, ou PC
[D, IDX] — Indirect indexé; déplacement accumulateur D à partir de X, Y, SP, ou PC

Codage Machine

`dd` — Adresse directe 8-bits de $0000 à $00FF. (L'octet poids fort est supposé $00).
`ee` — Octet poids fort d'un déplacement constant 16-bits en adressage indexé.
`eb` — Post octet d'échange ou de transfert.
`ff` — 8 bits de poids faible d'un déplacement constant signé sur 9-bits en adressage indexé, ou octet poids faible d'un déplacement constant sur 16-bits en adressage indexé.
`hh` — Octet poids fort d'une adresse étendue 16-bits.
`ii` — Valeur immédiate 8-bits.
`jj` — Octet poids fort d'une valeur immédiate sur 16-bits.
`kk` — Octet poids faible d'une valeur immédiate sur 16-bits.
`lb` — Post octet de primitive de boucle (DBNE).
`ll` — Octet poids faible d'une adresse étendue 16-bits.
`mm` — Valeur d'un masque immédiat sur 8-bits pour instructions de manipulation de bits. Les bits à 1 indiquent les bits concernés.
`pg` — Numéro de page programme (banque) utilisée dans l'instruction CALL.
`qq` — Octet poids fort d'un déplacement relatif 16-bits pour les branchements longs.
`rr` — Déplacement relatif signé de $80 (–128) à $7F (.... 127).
Déplacement relatif à l'octet suivant l'octet de déplacement relatif, ou octet poids faible d'un déplacement relatif 16-bits pour les branchements longs.
`xb` — Post octet d'adressage indexé.

Codes de condition

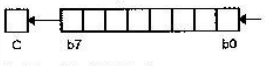
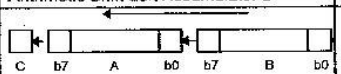
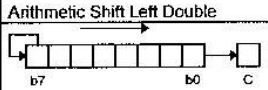
– — Bit d'état non affecté par l'opération.
0 — Bit d'état mis à 0 par l'opération.
1 — Bit d'état mis à 1 par l'opération.
Δ — Bit d'état affecté par l'opération.
↓ — Bit d'état peut être mis à 0 ou rester à 1, mais n'est pas mis à 1 par l'opération.
↑ — Bit d'état peut être mis à 1 ou rester à 0, mais n'est pas mis à 0 par l'opération.
? — Bit d'état peut être changé par l'opération, mais l'état final n'est pas défini.
! — Bit d'état utilisé de façon spéciale.

Source	Opération	Durée	Mode	Codage Machine	S X H I N Z V C
ABA	(A) + (B) ⇒ A Add Accumulators A and B	2	INH	18 06	––Δ– ΔΔΔΔ
ADCA #*opr8i* ADCA *opr8a* ADCA *opr16a* ADCA *oprx0_xysp* ADCA *oprx9,xysp* ADCA *oprx16,xysp* ADCA [D,*xysp*] ADCA [*oprx16,xysp*]	(A) + (M) + C ⇒ A Add with Carry to A	1 3 3 3 3 4 6 6	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	89 ii 99 dd B9 hh ll A9 xb A9 xb ff A9 xb ee ff A9 xb A9 xb ee ff	––Δ– ΔΔΔΔ
ADCB #*opr8i* ADCB *opr8a* ADCB *opr16a* ADCB *oprx0_xysp* ADCB *oprx9,xysp* ADCB *oprx16,xysp* ADCB [D,*xysp*] ADCB [*oprx16,xysp*]	(B) + (M) + C ⇒ B Add with Carry to B	1 3 3 3 3 4 6 6	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C9 ii D9 dd F9 hh ll E9 xb E9 xb ff E9 xb ee ff E9 xb E9 xb ee ff	––Δ– ΔΔΔΔ

ADDA #*opr8i*
ADDA *opr8a*
ADDA *opr16a*
ADDA *oprx0_xysp*
ADDA *oprx9,xysp*
ADDA *oprx16,xysp*
ADDA [D,*xysp*]
ADDA [*oprx16,xysp*]
	(A) + (M) ⇒ A
	Add without Carry to A
		1
		3
		3
		3
		3
		4
		6
		6
			IMM
			DIR
			EXT
			IDX
			IDX1
			IDX2
			[D,IDX]
			[IDX2
				8B ii
				9B dd
				BB hh ll
				AB xb
				AB xb ff
				AB xb ee ff
				AB xb
				AB xb ee ff
					––Δ– ΔΔΔΔ

ADDB #*opr8i*
ADDB *opr8a*
ADDB *opr16a*
ADDB *oprx0_xysp*
ADDB *oprx9,xysp*
ADDB *oprx16,xysp*
ADDB [D,*xysp*]
ADDB [*oprx16,xysp*]
	(B) + (M) ⇒ B
	Add without Carry to B
		1
		3
		3
		3
		3
		4
		6
		6
			IMM
			DIR
			EXT
			IDX
			IDX1
			IDX2
			[D,IDX]
			[IDX2]
				CB ii
				DB dd
				FB hh ll
				EB xb
				EB xb ff
				EB xb ee ff
				EB xb
				EB xb ee ff
					––Δ– ΔΔΔΔ

ADDD #*opr16i*
ADDD *opr8a*
ADDD *opr16a*
ADDD *oprx0_xysp*
ADDD *oprx9,xysp*
ADDD *oprx16,xysp*
ADDD [D,*xysp*]
ADDD [*oprx16,xysp*]
	(A:B) + (M:M+1) ⇒ A:B
	Add 16-Bit to D (A:B)
		2
		3
		3
		3
		3
		4
		6
		6
			IMM
			DIR
			EXT
			IDX
			IDX1
			IDX2
			[D,IDX]
			[IDX2]
				C3 jj kk
				D3 dd
				F3 hh ll
				E3 xb
				E3 xb ff
				E3 xb ee ff
				E3 xb
				E3 xb ee ff
					———— ΔΔΔΔ

ANDA #*opr8i* ANDA *opr8a* ANDA *opr16a* ANDA *oprx0_xysp* ANDA *oprx9,xysp* ANDA *oprx16,xysp* ANDA [D,*xysp*] ANDA [*oprx16,xysp*]	(A) • (M) ⇒A Logical AND A with Memory	1 3 3 3 3 4 6 6	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	84 ii 94 dd B4 hh ll A4 xb A4 xb ff A4 xb ee ff A4 xb A4 xb ee ff	——– ΔΔ0–
ANDB #*opr8i* ANDB *opr8a* ANDB *opr16a* ANDB *oprx0_xysp* ANDB *oprx9,xysp* ANDB *oprx16,xysp* ANDB [D,*xysp*] ANDB [*oprx16,xysp*]	(B) • (M) ⇒ B Logical AND B with Memory	1 3 3 3 3 4 6 6	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C4 ii D4 dd F4 hh ll E4 xb E4 xb ff E4 xb ee ff E4 xb E4 xb ee ff	——– ΔΔ0–
ANDCC #*opr8i*	(CCR) • (M) ⇒ CCR Logical AND CCR with Memory	1	IMM	10 ii	ΔΔΔΔ ΔΔΔΔ
ASL *opr16a* ASL *oprx0_xysp* ASL *oprx9,xysp* ASL *oprx16,xysp* ASL [D,*xysp*] ASL [*oprx16,xysp*] ASLA ASLB	 Arithmetic Shift Left Arithmetic Shift Left Accumulator A Arithmetic Shift Left Accumulator B	4 3 4 5 6 6 1 1	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	78 hh ll 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff 48 58	———— ΔΔΔΔ
ASLD	 Arithmetic Shift Left Double	1	INH	59	———— ΔΔΔΔ
ASR *opr16a* ASR *oprx0_xysp* ASR *oprx9,xysp* ASR *oprx16,xysp* ASR [D,*xysp*] ASR [*oprx16,xysp*] ASRA ASRB	 Arithmetic Shift Right Arithmetic Shift Right Accumulator A Arithmetic Shift Right Accumulator B	4 3 4 5 6 6 1 1	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	77 hh ll 67 xb 67 xb ff 67 xb ee ff 67 xb 67 xb ee ff 47 57	———— ΔΔΔΔ

Source	Opération	Durée	Mode	Codage Machine	S X H I N Z V C
BCC *rel8*	Branch if Carry Clear (C = 0)	3/1	REL	24 rr	---- ----
BCS *rel8*	Branch if Carry Set (C = 1)	3/1	REL	25 rr	---- ----
BEQ *rel8*	Branch if Equal (Z = 1)	3/1	REL	27 rr	---- ----
BGE *rel8*	Branch if Greater Than or Equal (N ⊕ V = 0) (signed)	3/1	REL	2C rr	---- ----
BGT *rel8*	Branch if Greater Than (Z + (N ⊕ V) = 0) (signed)	3/1	REL	2E rr	---- ----
BHI *rel8*	Branch if Higher (C + Z = 0) (unsigned)	3/1	REL	22 rr	---- ----
BHS *rel8*	Branch if Higher or Same (C = 0) (unsigned) Same as BCC	3/1	REL	24 rr	---- ----
BITA *#opr8i* BITA *opr8a* BITA *opr16a* BITA *oprx0_xysp* BITA *oprx9,xysp* BITA *oprx16,xysp* BITA [D,*xysp*] BITA [*oprx16,xysp*]	(A) • (M) Logical AND A with Memory Does not change Accumulator or Memory	1 3 3 3 3 4 6 6	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	85 ii 95 dd B5 hh ll A5 xb A5 xb ff A5 xb ee ff A5 xb A5 xb ee ff	---- ΔΔ0-
BITB *#opr8i* BITB *opr8a* BITB *opr16a* BITB *oprx0_xysp* BITB *oprx9,xysp* BITB *oprx16,xysp* BITB [D,*xysp*] BITB [*oprx16,xysp*]	(B) • (M) Logical AND B with Memory Does not change Accumulator or Memory	1 3 3 3 3 4 6 6	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C5 ii D5 dd F5 hh ll E5 xb E5 xb ff E5 xb ee ff E5 xb E5 xb ee ff	---- ΔΔ0-
BLE *rel8*	Branch if Less Than or Equal (Z + (N ⊕ V) =1) (signed)	3/1	REL	2F rr	---- ----
BLO *rel8*	Branch if Lower (C=1) (unsigned) Same as BCS	3/1	REL	25 rr	---- ----
BLS *rel8*	Branch if Lower or Same (C + Z = 1) (unsigned)	3/1	REL	23 rr	---- ----
BLT *rel8*	Branch if Less Than (N ⊕ V = 1) (signed)	3/1	REL	2D rr	---- ----
BMI *rel8*	Branch if Minus (N = 1)	3/1	REL	2B rr	---- ----
BNE *rel8*	Branch if Not Equal (Z = 0)	3/1	REL	26 rr	---- ----
BPL *rel8*	Branch if Plus (N = 0)	3/1	REL	2A rr	---- ----
BRA *rel8*	Branch Always (1 = 1)	3/1	REL	20 rr	---- ----
BRN *rel8*	Branch Never (if 1 = 0)	3/1	REL	21 rr	---- ----
BSR *rel8*	(SP) – 2 ⇒SP; RTN H :RTN L⇒M (SP) :M (SP+1) Subroutine address ⇒PC Branch to Subroutine	4	REL	07 rr	---- ----
BVC *rel8*	Branch if Overflow Bit Clear (V = 0)	3/1	REL	28 rr	---- ----
BVS *rel8*	Branch if Overflow Bit Set (V = 1)	3/1	REL	29 rr	---- ----
CALL *opr16a, page* CALL *oprx0_xysp, page* CALL *oprx9,xysp, page* CALL *oprx16,xysp, page* CALL [D,*xysp*] CALL [*oprx16, xysp*]	(SP) – 2 ⇒SP; RTN H :RTN L ⇒M (SP) :M (SP+1) (SP) – 1 ⇒SP; (PPG) ⇒M (SP) ; pg ⇒PPAGE register; Program address ⇒ PC Call subroutine in extended memory (Program may be located on another expansion memory page.) Indirect modes get program address and new pg value based on pointer.	7 7 7 8 10 10	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	4A hh ll pg 4B xb pg 4B xb ff pg 4B xb ee ff pg 4B xb 4B xb ee ff	---- ----
CBA	(A) – (B) Compare 8-Bit Accumulators Does not change Accumulators	2	INH	18 17	---- ΔΔΔΔ
CLR *opr16a* CLR *oprx0_xysp* CLR *oprx9,xysp* CLR *oprx16,xysp* CLR [*D,xysp*] CLR [*oprx16,xysp*] CLRA CLRB	0 ⇒ M Clear Memory Location 0 ⇒ A Clear Accumulator A 0 ⇒ B Clear Accumulator B	3 2 3 3 4 4 1 1	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	79 hh ll 69 xb 69 xb ff 69 xb ee ff 69 xb 69 xb ee ff 87 C7	---- 0100

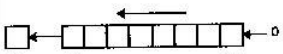
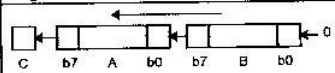
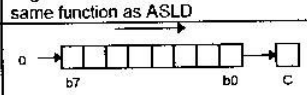
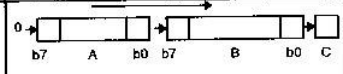
CMPA *#opr8i*
CMPA *opr8a*
CMPA *opr16a*
CMPA *oprx0_xysp*
CMPA *oprx9,xysp*
CMPA *oprx16,xysp*
CMPA [D,*xysp*]
CMPA [*oprx16,xysp*]
	(A) – (M)
	Compare Accumulator A with Memory
	Does not change Accumulator or Memory
		1
		3
		3
		3
		3
		4
		6
		6
			IMM
			DIR
			EXT
			IDX
			IDX1
			IDX2
			[D,IDX]
			[IDX2]
				81 ii
				91 dd
				B1 hh ll
				A1 xb
				A1 xb ff
				A1 xb ee ff
				A1 xb
				A1 xb ee ff
					---- ΔΔΔΔ

CMPB *#opr8i*
CMPB *opr8a*
CMPB *opr16a*
CMPB *oprx0_xysp*
CMPB *oprx9,xysp*
CMPB *oprx16,xysp*
CMPB [D,*xysp*]
CMPB [*oprx16,xysp*]
	(B) – (M)
	Compare Accumulator B with Memory
	Does not change Accumulator or Memory
		1
		3
		3
		3
		3
		4
		6
		6
			IMM
			DIR
			EXT
			IDX
			IDX1
			IDX2
			[D,IDX]
			[IDX2]
				C1 ii
				D1 dd
				F1 hh ll
				E1 xb
				E1 xb ff
				E1 xb ee ff
				E1 xb
				E1 xb ee ff
					---- ΔΔΔΔ

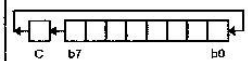
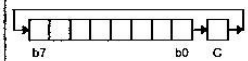
Source	Opération	Durée	Mode	Codage Machine	S X H I N Z V C
COM *opr16a*	(M) ⇒ M *equivalent to* $FF – (M) ⇒ M	4	EXT	71 hh ll	---- ΔΔ01
COM *oprx0_xysp*	1's Complement Memory Location	3	IDX	61 xb	
COM *oprx9,xysp*		4	IDX1	61 xb ff	
COM *oprx16,xysp*		5	IDX2	61 xb ee ff	
COM [D,*xysp*]		6	[D,IDX]	61 xb	
COM *[oprx16,xysp]*		6	[IDX2]	61 xb ee ff	
COMA	(A) ⇒ A Complement Accumulator A	1	INH	41	
COMB	(B) ⇒ B Complement Accumulator B	1	INH	51	
CPD *#opr16i*	(A:B) – (M:M+1)	2	IMM	8C jj kk	---- ΔΔΔΔ
CPD *opr8a*	Compare D to Memory (16-Bit)	3	DIR	9C dd	
CPD *opr16a*	Does not change Accumulator	3	EXT	BC hh ll	
CPD *oprx0_xysp*	or Memory	3	IDX	AC xb	
CPD *oprx9,xysp*		3	IDX1	AC xb ff	
CPD *oprx16,xysp*		4	IDX2	AC xb ee ff	
CPD [D,*xysp*]		6	[D,IDX]	AC xb	
CPD *[oprx16,xysp]*		6	[IDX2]	AC xb ee ff	
CPS *#opr16i*	(SP) – (M:M+1)	2	IMM	8F jj kk	---- ΔΔΔΔ
CPS *opr8a*	Compare SP to Memory (16-Bit)	3	DIR	9F dd	
CPS *opr16a*	Does not change Accumulator	3	EXT	BF hh ll	
CPS *oprx0_xysp*	or Memory	3	IDX	AF xb	
CPS *oprx9,xysp*		3	IDX1	AF xb ff	
CPS *oprx16,xysp*		4	IDX2	AF xb ee ff	
CPS [D,*xysp*]		6	[D,IDX]	AF xb	
CPS *[oprx16,xysp]*		6	[IDX2]	AF xb ee ff	
CPX *#opr16i*	(X) – (M:M+1)	2	IMM	8E jj kk	---- ΔΔΔΔ
CPX *opr8a*	Compare X to Memory (16-Bit)	3	DIR	9E dd	
CPX *opr16a*	Does not change Accumulator	3	EXT	BE hh ll	
CPX *oprx0_xysp*	or Memory	3	IDX	AE xb	
CPX *oprx9,xysp*		3	IDX1	AE xb ff	
CPX *oprx16,xysp*		4	IDX2	AE xb ee ff	
CPX [D,*xysp*]		6	[D,IDX]	AE xb	
CPX *[oprx16,xysp]*		6	[IDX2]	AE xb ee ff	
CPY *#opr16i*	(Y) – (M:M+1)	2	IMM	8D jj kk	-- -- ΔΔΔΔ
CPY *opr8a*	Compare Y to Memory (16-Bit)	3	DIR	9D dd	
CPY *opr16a*	Does not change Accumulator	3	EXT	BD hh ll	
CPY *oprx0_xysp*	or Memory	3	IDX	AD xb	
CPY *oprx9,xysp*		3	IDX1	AD xb ff	
CPY *oprx16,xysp*		4	IDX2	AD xb ee ff	
CPY [D,*xysp*]		6	[D,IDX]	AD xb	
CPY *[oprx16,xysp]*		6	[IDX2]	AD xb ee ff	
DEC *opr16a*	(M) – $01 ⇒ M	4	EXT	73 hh ll	--- ΔΔΔ–
DEC *oprx0_xysp*	Decrement Memory Location	3	IDX	63 xb	
DEC *oprx9,xysp*		4	IDX1	63 xb ff	
DEC *oprx16,xysp*		5	IDX2	63 xb ee ff	
DEC [D,*xysp*]		6	[D,IDX]	63 xb	
DEC *[oprx16,xysp]*		6	[IDX2]	63 xb ee ff	
DECA	(A) – $01 ⇒ A Decrement A	1	INH	43	
DECB	(B) – $01 ⇒ B Decrement B	1	INH	53	
DEX	(X) – $0001 ⇒ X Decrement Ind Reg X	1	INH	09	---- –Δ--
DEY	(Y) – $0001 ⇒ Y Decrement Ind Reg Y	1	INH	03	---- –Δ--
EDIV	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D	11	INH	11	---- ΔΔΔΔ
	32 by 16 Bit .16 Bit Divide (unsigned)				
EDIVS	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D	12	INH	18 14	---- ΔΔΔΔ
	32 by 16 Bit .16 Bit Divide (signed)				
EMAXD *oprx0_xysp*	MAX((D), (M:M+1)) ⇒ D	4	IDX	18 1A xb	---- ΔΔΔΔ
EMAXD *oprx9,xysp*	MAX of 2 Unsigned 16-Bit Values	4	IDX1	18 1A xb ff	
EMAXD *oprx16,xysp*	N, Z, V and C status bits reflect result of	5	IDX2	18 1A xb ee ff	
EMAXD [D,*xysp*]	internal compare ((D) – (M:M+1))	7	[D,IDX]	18 1A xb	
EMAXD *[oprx16,xysp]*		7	[IDX2]	18 1A xb ee ff	
EMAXM *oprx0_xysp*	MAX((D), (M:M+1)) ⇒ M:M+1	4	IDX	18 1E xb	---- ΔΔΔΔ
EMAXM *oprx9,xysp*	MAX of 2 Unsigned 16-Bit Values	5	IDX1	18 1E xb ff	
EMAXM *oprx16,xysp*	N, Z, V and C status bits reflect result of	6	IDX2	18 1E xb ee ff	
EMAXM [D,*xysp*]	internal compare ((D) – (M:M+1))	7	[D,IDX]	18 1E xb	
EMAXM *[oprx16,xysp]*		7	[IDX2]	18 1E xb ee ff	
EMIND *oprx0_xysp*	MIN((D), (M:M+1)) ⇒ D	4	IDX	18 1B xb	---- ΔΔΔΔ
EMIND *oprx9,xysp*	MIN of 2 Unsigned 16-Bit Values	4	IDX1	18 1B xb ff	
EMIND *oprx16,xysp*	N, Z, V and C status bits reflect result of	5	IDX2	18 1B xb ee ff	
EMIND [D,*xysp*]	internal compare ((D) – (M:M+1))	7	[D,IDX]	18 1B xb	
EMIND *[oprx16,xysp]*		7	[IDX2]	18 1B xb ee ff	
EMINM *oprx0_xysp*	MIN((D), (M:M+1)) ⇒ M:M+1	4	IDX	18 1F xb	---- ΔΔΔΔ
EMINM *oprx9,xysp*	MIN of 2 Unsigned 16-Bit Values	5	IDX1	18 1F xb ff	
EMINM *oprx16,xysp*	N, Z, V and C status bits reflect result of	6	IDX2	18 1F xb ee ff	
EMINM [D,*xysp*]	internal compare ((D) – (M:M+1))	7	[D,IDX]	18 1F xb	
EMINM *[oprx16,xysp]*		7	[IDX2]	18 1F xb ee ff	
EMUL	(D) × (Y) ⇒ Y:D	3	INH	13	---- ΔΔ–Δ
	16 by 16 Bit Multiply (unsigned)				
EMULS	(D) × (Y) ⇒ Y:D	3	INH	18 13	---- ΔΔ–Δ
	16 by 16 Bit Multiply (signed)				

Source	Opération	Durée	Mode	Codage Machine	S X H I N Z V C
EORA *#opr8i* EORA *opr8a* EORA *opr16a* EORA *oprx0_xysp* EORA *oprx9,xysp* EORA *oprx16,xysp* EORA [D,*xysp*] EORA [*oprx16,xysp*]	(A) ⊕ (M) ⇒ A Exclusive-OR A with Memory	1 3 3 3 3 4 6 6	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	88 ii 98 dd B8 hh ll A8 xb A8 xb ff A8 xb ee ff A8 xb A8 xb ee ff	---- ΔΔ0-
EORB *#opr8i* EORB *opr8a* EORB *opr16a* EORB *oprx0_xysp* EORB *oprx9,xysp* EORB *oprx16,xysp* EORB [D,*xysp*] EORB [*oprx16,xysp*]	(B) ⊕ (M) ⇒ B Exclusive-OR B with Memory	1 3 3 3 3 4 6 6	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C8 ii D8 dd F8 hh ll E8 xb E8 xb ff E8 xb ee ff E8 xb E8 xb ee ff	---- ΔΔ0-
EXG *abcdxys,abcdxys*	(r1) ⇔ (r2) (if r1 and r2 same size) *or* $00:(r1) ⇒ r2 (if r1=8-bit; r2=16-bit) *or* (r1 low) ⇔ (r2) (if r1=16-bit; r2=8-bit) *r1* and *r2* may be A, B, CCR, D, X, Y, or SP	1	INH	B7	---- ----
FDIV	(D) ÷ (X) ⇒ X; Remainder ⇒ D 16 by 16 Bit Fractional Divide	12	INH	18 11	---- -ΔΔΔ
IDIV	(D) ÷ (X) ⇒ X; Remainder ⇒ D 16 by 16 Bit Integer Divide (unsigned)	12	INH	18 15	---- ΔΔΔΔ
INC *opr16a* INC *oprx0_xysp* INC *oprx9,xysp* INC *oprx16,xysp* INC [D,*xysp*] INC [*oprx16,xysp*] INCA INCB	(M) + $01 ⇒ M Increment Memory Byte (A) + $01 ⇒ A Increment Acc. A (B) + $01 ⇒ B Increment Acc. B	4 3 4 5 6 6 1 1	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	72 hh ll 62 xb 62 xb ff 62 xb ee ff 62 xb 62 xb ee ff 42 52	---- ΔΔΔ-
INX	(X) + $0001 ⇒ X Increment Index Register X	1	INH	08	---- -Δ--
INY	(Y) + $0001 ⇒ Y Increment Index Register Y	1	INH	02	---- -Δ--
JMP *opr16a* JMP *oprx0_xysp* JMP *oprx9,xysp* JMP *oprx16,xysp* JMP [D,*xysp*] JMP [*oprx16,xysp*]	Routine address ⇒ PC Jump	3 3 3 4 6 6	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	06 hh ll 05 xb 05 xb ff 05 xb ee ff 05 xb 05 xb ee ff	---- ----
JSR *opr8a* JSR *opr16a* JSR *oprx0_xysp* JSR *oprx9,xysp* JSR *oprx16,xysp* JSR [D,*xysp*] JSR [*oprx16,xysp*]	(SP) – 2 ⇒ SP; RTN H :RTN L ⇒ M (SP) :M (SP+1) ; Subroutine address ⇒ PC Jump to Subroutine	4 4 4 4 5 7 7	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	17 dd 16 hh ll 15 xb 15 xb ff 15 xb ee ff 15 xb 15 xb ee ff	---- ----
LBCC *rel16*	Long Branch if Carry Clear (C = 0)	4/3	REL	18 24 qq rr	---- ----
LBCS *rel16*	Long Branch if Carry Set (C = 1)	4/3	REL	18 25 qq rr	---- ----
LBEQ *rel16*	Long Branch if Equal (Z = 1)	4/3	REL	18 27 qq rr	---- ----
LBGE *rel16*	Long Branch Greater Than or Equal (N ⊕ V = 0) (signed)	4/3	REL	18 2C qq rr	---- ----
LBGT *rel16*	Long Branch if Greater Than (Z ┼(N ⊕ V) = 0) (signed)	4/3	REL	18 2E qq rr	---- ----
LBHI *rel16*	Long Branch if Higher (C ┼Z = 0) (unsigned)	4/3	REL	18 22 qq rr	---- ----
LBHS *rel16*	Long Branch if Higher or Same (C = 0) (unsigned). Same function as LBCC	4/3	REL	18 24 qq rr	---- ----
LBLE *rel16*	Long Branch if Less Than or Equal (Z ┼ (N ⊕ V) = 1) (signed)	4/3	REL	18 2F qq rr	---- ----
LBLO *rel16*	Long Branch if Lower (C = 1) (unsigned). Same function as LBCS	4/3	REL	18 25 qq rr	---- ----
LBLS *rel16*	Long Branch if Lower or Same (C ┼ Z = 1) (unsigned)	4/3	REL	18 23 qq rr	---- ----
LBLT *rel16*	Long Branch if Less Than (N ⊕ V = 1) (signed)	4/3	REL	18 2D qq rr	---- ----
LBMI *rel16*	Long Branch if Minus (N = 1)	4/3	REL	18 2B qq rr	---- ----
LBNE *rel16*	Long Branch if Not Equal (Z = 0)	4/3	REL	18 26 qq rr	---- ----
LBPL *rel16*	Long Branch if Plus (N = 0)	4/3	REL	18 2A qq rr	---- ----
LBRA *rel16*	Long Branch Always (1 = 1)	4	REL	18 20 qq rr	---- ----
LBRN *rel16*	Long Branch Never (1 = 0)	3	REL	18 21 qq rr	---- ----
LBVC *rel16*	Long Branch if Overflow Bit Clear (V=0)	4/3	REL	18 28 qq rr	---- ----
LBVS *rel16*	Long Branch if Overflow Bit Set (V = 1)	4/3	REL	18 29 qq rr	---- ----

LDAA *#opr8i*
LDAA *opr8a*
LDAA *opr16a*
LDAA *oprx0_xysp*
LDAA *oprx9,xysp*
LDAA *oprx16,xysp*
LDAA [D,*xysp*]
LDAA [*oprx16,xysp*]
	(M) ⇒ A
	Load Accumulator A
		1
		3
		3
		3
		3
		4
		6
		6
			IMM
			DIR
			EXT
			IDX
			IDX1
			IDX2
			[D,IDX]
			[IDX2]
				86 ii
				96 dd
				B6 hh ll
				A6 xb
				A6 xb ff
				A6 xb ee ff
				A6 xb
				A6 xb ee ff
					---- ΔΔ0-

Source	Opération	Durée	Mode	Codage Machine	S X H I N Z V C
LDAB #*opr8i*	(M) ⇒ B	1	IMM	C6 ii	---- ΔΔ0-
LDAB *opr8a*	Load Accumulator B	3	DIR	D6 dd	
LDAB *opr16a*		3	EXT	F6 hh ll	
LDAB *oprx0_xysp*		3	IDX	E6 xb	
LDAB *oprx9,xysp*		3	IDX1	E6 xb ff	
LDAB *oprx16,xysp*		4	IDX2	E6 xb ee ff	
LDAB [D,*xysp*]		6	[D,IDX]	E6 xb	
LDAB [*oprx16,xysp*]		6	[IDX2]	E6 xb ee ff	
LDD #*opr16i*	(M:M+1) ⇒ A:B	2	IMM	CC jj kk	---- ΔΔ0-
LDD *opr8a*	Load Double Accumulator D	3	DIR	DC dd	
LDD *opr16a*		3	EXT	FC hh ll	
LDD *oprx0_xysp*		3	IDX	EC xb	
LDD *oprx9,xysp*		3	IDX1	EC xb ff	
LDD *oprx16,xysp*		4	IDX2	EC xb ee ff	
LDD [D,*xysp*]		6	[D,IDX]	EC xb	
LDD [*oprx16,xysp*]		6	[IDX2]	EC xb ee ff	
LDS #*opr16i*	(M:M+1) ⇒ SP	2	IMM	CF jj kk	---- ΔΔ0-
LDS *opr8a*	Load Stack Pointer	3	DIR	DF dd	
LDS *opr16a*		3	EXT	FF hh ll	
LDS *oprx0_xysp*		3	IDX	EF xb	
LDS *oprx9,xysp*		3	IDX1	EF xb ff	
LDS *oprx16,xysp*		4	IDX2	EF xb ee ff	
LDS [D,*xysp*]		6	[D,IDX]	EF xb	
LDS [*oprx16,xysp*]		6	[IDX2]	EF xb ee ff	
LDX #*opr16i*	(M:M+1) ⇒ X	2	IMM	CE jj kk	---- ΔΔ0-
LDX *opr8a*	Load Index Register X	3	DIR	DE dd	
LDX *opr16a*		3	EXT	FE hh ll	
LDX *oprx0_xysp*		3	IDX	EE xb	
LDX *oprx9,xysp*		3	IDX1	EE xb ff	
LDX *oprx16,xysp*		4	IDX2	EE xb ee ff	
LDX [D,*xysp*]		6	[D,IDX]	EE xb	
LDX [*oprx16,xysp*]		6	[IDX2]	EE xb ee ff	
LDY #*opr16i*	(M:M+1) ⇒ Y	2	IMM	CD jj kk	---- ΔΔ0-
LDY *opr8a*	Load Index Register Y	3	DIR	DD dd	
LDY *opr16a*		3	EXT	FD hh ll	
LDY *oprx0_xysp*		3	IDX	ED xb	
LDY *oprx9,xysp*		3	IDX1	ED xb ff	
LDY *oprx16,xysp*		4	IDX2	ED xb ee ff	
LDY [D,*xysp*]		6	[D,IDX]	ED xb	
LDY [*oprx16,xysp*]		6	[IDX2]	ED xb ee ff	
LEAS *oprx0_xysp*	Effective Address ⇒ SP	2	IDX	1B xb	---- ----
LEAS *oprx9,xysp*	Load Effective Address into SP	2	IDX1	1B xb ff	
LEAS *oprx16,xysp*		2	IDX2	1B xb ee ff	
LEAX *oprx0_xysp*	Effective Address ⇒ X	2	IDX	1A xb	---- ----
LEAX *oprx9,xysp*	Load Effective Address into X	2	IDX1	1A xb ff	
LEAX *oprx16,xysp*		2	IDX2	1A xb ee ff	
LEAY *oprx0_xysp*	Effective Address ⇒ Y	2	IDX	19 xb	---- ----
LEAY *oprx9,xysp*	Load Effective Address into Y	2	IDX1	19 xb ff	
LEAY *oprx16,xysp*		2	IDX2	19 xb ee ff	
LSL *opr16a*		4	EXT	78 hh ll	---- ΔΔΔΔ
LSL *oprx0_xysp*		3	IDX	68 xb	
LSL *oprx9,xysp*		4	IDX1	68 xb ff	
LSL *oprx16,xysp*		5	IDX2	68 xb ee ff	
LSL [D,*xysp*]	Logical Shift Left. same function as ASL	6	[D,IDX]	68 xb	
LSL [*oprx16,xysp*]		6	[IDX2]	68 xb ee ff	
LSLA	Logical Shift Accumulator A to Left	1	INH	48	
LSLB	Logical Shift Accumulator B to Left	1	INH	58	
LSLD		1	INH	59	---- ΔΔΔΔ
	Logical Shift Left D Accumulator same function as ASLD				
LSR *opr16a*		4	EXT	74 hh ll	---- 0ΔΔΔ
LSR *oprx0_xysp*		3	IDX	64 xb	
LSR *oprx9,xysp*		4	IDX1	64 xb ff	
LSR *oprx16,xysp*		5	IDX2	64 xb ee ff	
LSR [D,*xysp*]	Logical Shift Right	6	[D,IDX]	64 xb	
LSR [*oprx16,xysp*]		6	[IDX2]	64 xb ee ff	
LSRA	Logical Shift Accumulator A to Right	1	INH	44	
LSRB	Logical Shift Accumulator B to Right	1	INH	54	
LSRD		1	INH	49	---- 0ΔΔΔ
	Logical Shift Right D Accumulator				
MAXA *oprx0_xysp*	MAX((A), (M)) ⇒ A	4	IDX	18 18 xb	---- ΔΔΔΔ
MAXA *oprx9,xysp*	MAX of 2 Unsigned 8-Bit Values	4	IDX1	18 18 xb ff	
MAXA *oprx16,xysp*	N, Z, V and C status bits reflect result of	5	IDX2	18 18 xb ee ff	
MAXA [D,*xysp*]	internal compare	7	[D,IDX]	18 18 xb	
MAXA [*oprx16,xysp*]		7	[IDX2]	18 18 xb ee ff	

Source	Opération	Durée	Mode	Codage Machine	S X H I N Z V C
MAXM *oprx0_xysp* MAXM *oprx9,xysp* MAXM *oprx16,xysp* MAXM [D,*xysp*] MAXM *[oprx16,xysp]*	MAX((A), (M)) ⇒ M MAX of 2 Unsigned 8-Bit Values N, Z, V and C status bits reflect result of internal compare ((A) – (M)).	4 5 6 7 7	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1C xb 18 1C xb ff 18 1C xb ee ff 18 1C xb 18 1C xb ee ff	---- ΔΔΔΔ
MINA *oprx0_xysp* MINA *oprx9,xysp* MINA *oprx16,xysp* MINA [D,*xysp*] MINA *[oprx16,xysp]*	MIN((A), (M)) ⇒ A MIN of 2 Unsigned 8-Bit Values N, Z, V and C status bits reflect result of internal compare ((A) – (M)).	4 4 5 7 7	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 19 xb 18 19 xb ff 18 19 xb ee ff 18 19 xb 18 19 xb ee ff	---- ΔΔΔΔ
MINM *oprx0_xysp* MINM *oprx9,xysp* MINM *oprx16,xysp* MINM [D,*xysp*] MINM *[oprx16,xysp]*	MIN((A), (M)) ⇒ M MIN of 2 Unsigned 8-Bit Values N, Z, V and C status bits reflect result of internal compare ((A) – (M)).	4 5 6 7 7	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1D xb 18 1D xb ff 18 1D xb ee ff 18 1D xb 18 1D xb ee ff	---- ΔΔΔΔ
MOVB *#opr8, opr16a* 1 MOVB *#opr8i, oprx0_xysp* 1 MOVB *opr16a, opr16a* 1 MOVB *opr16a, oprx0_xysp* 1 MOVB *oprx0_xysp, opr16a* 1 MOVB *oprx0_xysp, oprx0_xysp* 1	(M 1) ⇒ M 2 Memory to Memory Byte-Move (8-Bit)	4 4 6 5 5 5	IMM-EXT IMM-IDX EXT-EXT EXT-IDX IDX-EXT IDX-IDX	18 0B ii hh ll 18 08 xb ii 18 0C hh ll hh ll 18 09 xb hh ll 18 0D xb hh ll 18 0A xb xb	---- ----
MOVW *#oprx16, opr16a* 1 MOVW *#opr16i, oprx0_xysp* 1 MOVW *opr16a, opr16a* 1 MOVW *opr16a, oprx0_xysp* 1 MOVW *oprx0_xysp, opr16a* 1 MOVW *oprx0_xysp, oprx0_xysp* 1	(M:M+1 1) ⇒ M:M+1 2 Memory to Memory Word-Move (16-Bit)	5 4 6 5 5 5	IMM-EXT IMM-IDX EXT-EXT EXT-IDX IDX-EXT IDX-IDX	18 03 jj kk hh ll 18 00 xb jj kk 18 04 hh ll hh ll 18 01 xb hh ll 18 05 xb hh ll 18 02 xb xb	---- ----
MUL	(A) × (B) ⇒ A:B 8 by 8 Unsigned Multiply	1	INH	12	---- ----
NEG *opr16a* NEG *oprx0_xysp* NEG *oprx9,xysp* NEG *oprx16,xysp* NEG [D,*xysp*] NEG *[oprx16,xysp]* NEGA NEGB	0 – (M) ⇒ M *equivalent to* (M) + 1 ⇒ M Two's Complement Negate 0 – (A) ⇒ A *equivalent to* (A) + 1 ⇒ A Negate Accumulator A 0 – (B) ⇒ B *equivalent to* (B) + 1 ⇒ B Negate Accumulator B	4 3 4 5 7 7 1 1	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	70 hh ll 60 xb 60 xb ff 60 xb ee ff 60 xb 60 xb ee ff 40 50	---- ΔΔΔΔ
NOP	No Operation	1	INH	A7	---- ----
ORAA *#opr8i* ORAA *opr8a* ORAA *opr16a* ORAA *oprx0_xysp* ORAA *oprx9,xysp* ORAA *oprx16,xysp* ORAA [D,*xysp*] ORAA *[oprx16,xysp]*	(A) + (M) ⇒ A Logical OR A with Memory	1 3 3 3 3 4 6 6	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8A ii 9A dd BA hh ll AA xb AA xb ff AA xb ee ff AA xb AA xb ee ff	---- ΔΔ0-
ORAB *#opr8i* ORAB *opr8a* ORAB *opr16a* ORAB *oprx0_xysp* ORAB *oprx9,xysp* ORAB *oprx16,xysp* ORAB [D,*xysp*] ORAB *[oprx16,xysp]*	(B) + (M) ⇒ B Logical OR B with Memory	1 3 3 3 3 4 6 6	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	CA ii DA dd FA hh ll EA xb EA xb ff EA xb ee ff EA xb EA xb ee ff	---- ΔΔ0-
ORCC *#opr8i*	Logical OR CCR with Memory	1	IMM	14 ii	Δ-ΔΔΔΔΔ
PSHA	(SP) – 1 ⇒ SP; (A) ⇒ M (SP) Push Accumulator A onto Stack	2	INH	36	---- ----
PSHB	(SP) – 1 ⇒ SP; (B) ⇒ M (SP) Push Accumulator B onto Stack	2	INH	37	---- ----
PSHC	(SP) – 1 ⇒ SP; (CCR) ⇒ M (SP) Push CCR onto Stack	2	INH	39	---- ----
PSHD	(SP) – 2 ⇒ SP; (A:B) ⇒ M (SP) :M (SP+1) Push D Accumulator onto Stack	2	INH	3B	---- ----
PSHX	(SP) – 2 ⇒ SP; (X H :X L) ⇒ M (SP) :M (SP+1) Push Index Register X onto Stack	2	INH	34	---- ----
PSHY	(SP) – 2 ⇒ SP; (Y H :Y L) ⇒ M (SP) :M (SP+1) Push Index Register Y onto Stack	2	INH	35	---- ----
PULA	(M (SP)) ⇒ A; (SP) + 1 ⇒ SP Pull Accumulator A from Stack	3	INH	32	---- ----
PULB	(M (SP)) ⇒ B; (SP) + 1 ⇒ SP Pull Accumulator B from Stack	3	INH	33	---- ----
PULC	(M (SP)) ⇒ CCR; (SP) + 1 ⇒ SP Pull CCR from Stack	3	INH	38	---- ----
PULD	(M (SP) :M (SP+1)) ⇒ A:B; (SP) + 2 ⇒ SP Pull D from Stack	3	INH	3A	---- ----
PULX	(M (SP) :M (SP+1)) ⇒ X H :X L ;(SP) + 2 ⇒ SP Pull Index Register X from Stack	3	INH	30	---- ----
PULY	(M (SP) :M (SP+1)) ⇒ Y H :Y L ; (SP) + 2 ⇒ SP Pull Index Register Y from Stack	3	INH	31	---- ----

Source	Opération	Durée	Mode	Codage Machine	S X H I N Z V C
ROL *opr16a*		4	EXT	75 hh ll	---- ΔΔΔΔ
ROL *oprx0_xysp*		3	IDX	65 xb	
ROL *oprx9,xysp*		4	IDX1	65 xb ff	
ROL *oprx16,xysp*		5	IDX2	65 xb ee ff	
ROL [D,*xysp*]	Rotate Memory Left through Carry	7	[D,IDX]	65 xb	
ROL [*oprx16,xysp*]		7	[IDX2]	65 xb ee ff	
ROLA	Rotate A Left through Carry	1	INH	45	
ROLB	Rotate B Left through Carry	1	INH	55	
ROR *opr16a*		4	EXT	76 hh ll	---- ΔΔΔΔ
ROR *oprx0_xysp*		3	IDX	66 xb	
ROR *oprx9,xysp*		4	IDX1	66 xb ff	
ROR *oprx16,xysp*		5	IDX2	66 xb ee ff	
ROR [D,*xysp*]	Rotate Memory Right through Carry	7	[D,IDX]	66 xb	
ROR [*oprx16,xysp*]		7	[IDX2]	66 xb ee ff	
RORA	Rotate A Right through Carry	1	INH	46	
RORB	Rotate B Right through Carry	1	INH	56	
RTC	(M (SP)) ⇒ PPAGE; (SP) + 1 ⇒ SP; (M (SP) :M (SP+1)) ⇒ PC H :PC L ; (SP) + 2 ⇒ SP Return from Call	7	INH	0A	---- ----
RTI	(M (SP)) ⇒ CCR; (SP) + 1 ⇒ SP (M (SP) :M (SP+1)) ⇒ B:A; (SP) + 2 ⇒ SP (M (SP) :M (SP+1)) ⇒ X H :X L ; (SP) + 4 ⇒ SP (M (SP) :M (SP+1)) ⇒ PC H :PC L ; (SP) – 2 ⇒ SP (M (SP) :M (SP+1)) ⇒ Y H :Y L ; (SP) + 4 ⇒ SP Return from Interrupt	8/10	INH	0B	---- ----
RTS	(M (SP) :M (SP+1)) ⇒ PC H :PC L ; (SP) + 2 ⇒ SP Return from Subroutine	5	INH	3D	---- ----
SBA	(A) – (B) ⇒ A Subtract B from A	2	INH	18 16	---- ΔΔΔΔ
SBCA *#opr8i*	(A) – (M) – C ⇒ A	1	IMM	82 ii	---- ΔΔΔΔ
SBCA *opr8a*	Subtract with Borrow from A	3	DIR	92 dd	
SBCA *opr16a*		3	EXT	B2 hh ll	
SBCA *oprx0_xysp*		3	IDX	A2 xb	
SBCA *oprx9,xysp*		3	IDX1	A2 xb ff	
SBCA *oprx16,xysp*		4	IDX2	A2 xb ee ff	
SBCA [D,*xysp*]		6	[D,IDX]	A2 xb	
SBCA [*oprx16,xysp*]		6	[IDX2]	A2 xb ee ff	
SBCB *#opr8i*	(B) – (M) – C ⇒ B	1	IMM	C2 ii	---- ΔΔΔΔ
SBCB *opr8a*	Subtract with Borrow from B	3	DIR	D2 dd	
SBCB *opr16a*		3	EXT	F2 hh ll	
SBCB *oprx0_xysp*		3	IDX	E2 xb	
SBCB *oprx9,xysp*		3	IDX1	E2 xb ff	
SBCB *oprx16,xysp*		4	IDX2	E2 xb ee ff	
SBCB [D,*xysp*]		6	[D,IDX]	E2 xb	
SBCB [*oprx16,xysp*]		6	[IDX2]	E2 xb ee ff	
STAA *opr8a*	(A) ⇒ M	2	DIR	5A dd	---- ΔΔ0–
STAA *opr16a*	Store Accumulator A to Memory	3	EXT	7A hh ll	
STAA *oprx0_xysp*		2	IDX	6A xb	
STAA *oprx9,xysp*		3	IDX1	6A xb ff	
STAA *oprx16,xysp*		3	IDX2	6A xb ee ff	
STAA [D,*xysp*]		4	[D,IDX]	6A xb	
STAA [*oprx16,xysp*]		4	[IDX2]	6A xb ee ff	
STAB *opr8a*	(B) ⇒ M	2	DIR	5B dd	---- ΔΔ0–
STAB *opr16a*	Store Accumulator B to Memory	3	EXT	7B hh ll	
STAB *oprx0_xysp*		2	IDX	6B xb	
STAB *oprx9,xysp*		3	IDX1	6B xb ff	
STAB *oprx16,xysp*		3	IDX2	6B xb ee ff	
STAB [D,*xysp*]		4	[D,IDX]	6B xb	
STAB [*oprx16,xysp*]		4	[IDX2]	6B xb ee ff	
STD *opr8a*	(A) ⇒ M, (B) ⇒ M+1	2	DIR	5C dd	---- ΔΔ0–
STD *opr16a*	Store Double Accumulator	3	EXT	7C hh ll	
STD *oprx0_xysp*		2	IDX	6C xb	
STD *oprx9,xysp*		3	IDX1	6C xb ff	
STD *oprx16,xysp*		3	IDX2	6C xb ee ff	
STD [D,*xysp*]		4	[D,IDX]	6C xb	
STD [*oprx16,xysp*]		4	[IDX2]	6C xb ee ff	
STOP	(SP) – 2 ⇒ SP; RTN H :RTN L ⇒ M (SP) :M (SP+1) ; (SP) – 2 ⇒ P; (Y H :Y L) ⇒ M (SP) :M (SP+1) ; (SP) – 2 ⇒ SP; (X H :X L) ⇒ M (SP) :M (SP+1) ; (SP) – 2 ⇒ SP; (B:A) ⇒ M (SP) :M (SP+1) ; (SP) – 1 ⇒ SP; (CCR) ⇒ M (SP) ; STOP All Clocks Registers stacked to allow quicker recovery by interrupt. If S control bit = 1, the STOP instruction is disabled and acts like a two-cycle NOP	8/6/2	INH	18 3E	---- ----

Source	Opération	Durée	Mode	Codage Machine	S X H I N Z V C
STS *opr8a* STS *opr16a* STS *oprx0_xysp* STS *oprx9,xysp* STS *oprx16,xysp* STS [D,*xysp*] STS [*oprx16,xysp*]	(SP H :SP L) → :M+1 Store Stack Pointer	2 3 2 3 3 4 4	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5F dd 7F hh ll 6F xb 6F xb ff 6F xb ee ff 6F xb 6F xb ee ff	---- ΔΔ0-
STX *opr8a* STX *opr16a* STX *oprx0_xysp* STX *oprx9,xysp* STX *oprx16,xysp* STX [D,*xysp*] STX [*oprx16,xysp*]	(X H :X L) ⇒ M:M+1 Store Index Register X	2 3 2 3 3 4 4	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5E dd 7E hh ll 6E xb 6E xb ff 6E xb ee ff 6E xb 6E xb ee ff	---- ΔΔ0-
STY *opr8a* STY *opr16a* STY *oprx0_xysp* STY *oprx9,xysp* STY *oprx16,xysp* STY [D,*xysp*] STY [*oprx16,xysp*]	(Y H :Y L) ⇒ M:M+1 Store Index Register Y	2 3 2 3 3 4 4	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5D dd 7D hh ll 6D xb 6D xb ff 6D xb ee ff 6D xb 6D xb ee ff	---- ΔΔ0-

SUBA #*opr8i*
SUBA *opr8a*
SUBA *opr16a*
SUBA *oprx0_xysp*
SUBA *oprx9,xysp*
SUBA *oprx16,xysp*
SUBA [D,*xysp*]
SUBA [*oprx16,xysp*]
	(A) – (M) ⇒ A
	Subtract Memory from Accumulator A
		1
		3
		3
		3
		3
		4
		6
		6
			IMM
			DIR
			EXT
			IDX
			IDX1
			IDX2
			[D,IDX]
			[IDX2]
				80 ii
				90 dd
				B0 hh ll
				A0 xb
				A0 xb ff
				A0 xb ee ff
				A0 xb
				A0 xb ee ff
					---- ΔΔΔΔ

SUBB #*opr8i*
SUBB *opr8a*
SUBB *opr16a*
SUBB *oprx0_xysp*
SUBB *oprx9,xysp*
SUBB *oprx16,xysp*
SUBB [D,*xysp*]
SUBB [*oprx16,xysp*]
	(B) – (M) → B
	Subtract Memory from Accumulator B
		1
		3
		3
		3
		3
		4
		6
		6
			IMM
			DIR
			EXT
			IDX
			IDX1
			IDX2
			[D,IDX]
			[IDX2]
				C0 ii
				D0 dd
				F0 hh ll
				E0 xb
				E0 xb ff
				E0 xb ee ff
				E0 xb
				E0 xb ee ff
					---- ΔΔΔΔ

SUBD #*oprl6i*
SUBD *opr8a*
SUBD *opr16a*
SUBD *oprx0_xysp*
SUBD *oprx9,xysp*
SUBD *oprx16,xysp*
SUBD [D,*xysp*]
SUBD [*oprx16,xysp*]
	(D) – (M:M+1) ⇒ D
	Subtract Memory from D (A:B)
		2
		3
		3
		3
		3
		4
		6
		6
			IMM
			DIR
			EXT
			IDX
			IDX1
			IDX2
			[D,IDX]
			[IDX2]
				83 jj kk
				93 dd
				B3 hh ll
				A3 xb
				A3 xb ff
				A3 xb ee ff
				A3 xb
				A3 xb ee ff
					---- ΔΔΔΔ

SWI	(SP) – 2 ⇒ SP; RTN H :RTN L ⇒ M (SP) :M (SP+1) ; (SP) –2 ⇒ SP; (Y H :Y L) ⇒ M (SP) :M (SP+1) ; (SP) –2 ⇒ P; (X H :X L) ⇒ M (SP) :M (SP+1) ; (SP) –2 ⇒ SP; (B:A) ⇒ M (SP) :M (SP+1) ; (SP) – 1 ⇒ SP; (CCR) ⇒ M (SP) 1 ⇒ I; (SWI Vector) → PC Software Interrupt	9/5	INH	3F	11-1 ----
TAB	(A) ⇒ B Transfer A to B	2	INH	18 0E	---- ΔΔ0-
TBA	(B) ⇒ A Transfer B to A	2	INH	18 0F	---- ΔΔ0-
TFR *abcdxys,abcdxys*	(r1) ⇒ r2 *or* $00:(r1) ⇒ r2 *or* (r1[7:0]) ⇒ r2 Transfer Register to Register r1 and r2 may be A, B, CCR, D, X, Y, or SP	1	INH	B7 eb	---- ----
TST *opr16a* TST *oprx0_xysp* TST *oprx9,xysp* TST *oprx16,xysp* TST [D,*xysp*] TST [*oprx16,xysp*] TSTA TSTB	(M) – 0 Test Memory for Zero or Minus (A) – 0 Test A for Zero or Minus (B) – 0 Test B for Zero or Minus	3 3 3 4 6 6 1 1	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	F7 hh ll E7 xb E7 xb ff E7 xb ee ff E7 xb E7 xb ee ff 97 D7	---- ΔΔ00
WAI	(SP) –2 ⇒ SP; RTN H :RTN L ⇒ M (SP) :M (SP+1) ; (SP) –2 ⇒ SP; (Y H :Y L) ⇒ M (SP) :M (SP+1) ; (SP) –2 ⇒ SP; (X H :X L) ⇒ M (SP) :M (SP+1) ; (SP) –2 ⇒ SP; (B:A) ⇒ M (SP) :M (SP+1) ; (SP) – 1 ⇒ SP; (CCR) ⇒ M (SP) ; WAIT for interrupt	7/6	INH	3E	---1 .----

VECTEURS D'INTERRUPTION

Vect.Ad	Interrupt Source	Mask	Local Enable	HPRIO
FFFE,FFFF	Reset	None	None	–
FFFC, FFFD	Clock Monitor fail reset	None	PLLCTL (CME, SCME)	–
FFFA, FFFB	COP failure reset	None	COP rate select	–
FFF8, FFF9	Unimplemented instruction trap	None	None	–
FFF6, FFF7	SWI	None	None	–
FFF4, FFF5	XIRQ	X-Bit	None	–
FFF2, FFF3	IRQ	I-Bit	IRQCR (IRQEN)	F2
FFF0, FFF1	Real Time Interrupt	I-Bit	CRGINT (RTIE)	F0
FFEE, FFEF	Enhanced Capture Timer channel 0	I-Bit	TIE (C0I)	EE
FFEC, FFED	Enhanced Capture Timer channel 1	I-Bit	TIE (C1I)	EC
FFEA, FFEB	Enhanced Capture Timer channel 2	I-Bit	TIE (C2I)	EA
FFE8, FFE9	Enhanced Capture Timer channel 3	I-Bit	TIE (C3I)	E8
FFE6, FFE7	Enhanced Capture Timer channel 4	I-Bit	TIE (C4I)	E6
FFE4, FFE5	Enhanced Capture Timer channel 5	I-Bit	TIE (C5I)	E4
FFE2, FFE3	Enhanced Capture Timer channel 6	I-Bit	TIE (C6I)	E2
FFE0, FFE1	Enhanced Capture Timer channel 7	I-Bit	TIE (C7I)	E0
FFDE, FFDF	Enhanced Capture Timer overflow	I-Bit	TSRC2 (TOF)	DE
FFDC,FFDD	Pulse accumulator A overflow	I-Bit	PACTL (PAOVI)	DC
FFDA, FFDB	Pulse accumulator input edge	I-Bit	PACTL (PAI)	DA
FFD8,FFD9	SPI0	I-Bit	SP0CR1 (SPIE, SPTIE)	D8
FFD6, FFD7	SCI0	I-Bit	SC0CR2 (TIE, TCIE, RIE, ILIE)	D6
FFD4, FFD5	SCI1	I-Bit	SC1CR2 (TIE, TCIE, RIE, ILIE)	D4
FFD2, FFD3	ATD0	I-Bit	ATD0CTL2 (ASCIE)	D2
FFD0, FFD1	ATD1	I-Bit	ATD1CTL2 (ASCIE)	D0
FFCE, FFCF	Port J	I-Bit	PTJIF (PTJIE)	CE
FFCC, FFCD	Port H	I-Bit	PTHIF(PTHIE)	CC
FFCA, FFCB	Modulus Down Counter underflow	I-Bit	MCCTL(MCZI)	CA
FFC8, FFC9	Pulse Accumulator B Overflow	I-Bit	PBCTL(PBOVI)	C8
FFC6, FFC7	CRGPLL lock	I-Bit	CRGINT(LOCKIE)	C6
FFC4, FFC5	CRG Self Clock Mode	I-Bit	CRGINT (SCMIE)	C4
FFC2, FFC3	BDLC	I-Bit	DLCBCR1(IE)	C2
FFC0, FFC1	IIC Bus	I-Bit	IBCR (IBIE)	C0
FFBE, FFBF	SPI1	I-Bit	SP1CR1 (SPIE, SPTIE)	BE
FFBC, FFBD	SPI2	I-Bit	SP2CR1 (SPIE, SPTIE)	BC
FFBA, FFBB	EEPROM	I-Bit	EECTL(CCIE, CBEIE)	BA
FFB8, FFB9	FLASH	I-Bit	FCTL(CCIE, CBEIE)	B8
FFB6, FFB7	CAN0 wake-up	I-Bit	CAN0RIER (WUPIE)	B6
FFB4, FFB5	CAN0 errors	I-Bit	CAN0RIER (CSCIE, OVRIE)	B4
FFB2, FFB3	CAN0 receive	I-Bit	CAN0RIER (RXFIE)	B2
FFB0, FFB1	CAN0 transmit	I-Bit	CAN0TIER (TXEIE2-TXEIE0)	B0
FFAE, FFAF	CAN1 wake-up	I-Bit	CAN1RIER (WUPIE)	AE
FFAC, FFAD	CAN1 errors	I-Bit	CAN1RIER (CSCIE, OVRIE)	AC
FFAA, FFAB	CAN1 receive	I-Bit	CAN1RIER (RXFIE)	AA
FFA8, FFA9	CAN1 transmit	I-Bit	CAN1TIER (TXEIE2-TXEIE0)	A8
FFA6, FFA7	CAN2 wake-up	I-Bit	CAN2RIER (WUPIE)	A6
FFA4, FFA5	CAN2 errors	I-Bit	CAN2RIER (CSCIE, OVRIE)	A4
FFA2, FFA3	CAN2 receive	I-Bit	CAN2RIER (RXFIE)	A2
FFA0, FFA1	CAN2 transmit	I-Bit	CAN2TIER (TXEIE2-TXEIE0)	A0
FF9E, FF9F	CAN3 wake-up	I-Bit	CAN3RIER (WUPIE)	9E
FF9C, FF9D	CAN3 errors	I-Bit	CAN3RIER (TXEIE2-TXEIE0)	9C
FF9A, FF9B	CAN3 receive	I-Bit	CAN3RIER (RXFIE)	9A
FF98, FF99	CAN3 transmit	I-Bit	CAN3TIER (TXEIE2-TXEIE0)	98
FF96, FF97	CAN4 wake-up	I-Bit	CAN4RIER (WUPIE)	96
FF94, FF95	CAN4 errors	I-Bit	CAN4RIER (CSCIE, OVRIE)	94
FF92, FF93	CAN4 receive	I-Bit	CAN4RIER (RXFIE)	92
FF90, FF91	CAN4 transmit	I-Bit	CAN4TIER (TXEIE2-TXEIE0)	90
FF8E, FF8F	Port P Interrupt	I-Bit	PTPIF (PTPIE)	8E
FF8C, FF8D	PWM Emergency Shutdown	I-Bit	PWMSDN (PWMIE)	8C
FF80 to FF8B	Reserved			

	CCR	ACCB	ACCA	IXH	IXL	IYH	IYL	PCH	PCL
SP-9	SP-8	SP-7	SP-6	SP-5	SP-4	SP-3	SP-2	SP-1	SP

Ordre d'empilement des registres sur interruption. SP = valeur avant inter

Moniteur D-BUG12

On trouvera une description complète du moniteur D-BUG12 et de ses commandes dans le fichier de documentation :

DB12RG4.PDF.

Ce moniteur est très général et peut être utilisé dans d'autres contextes que celui de nos TP. La liste des commandes qui suit est à peu près complète dans ce cadre. Au cas où elle présente des insuffisances, on se reportera sans hésiter au fichier de référence.

PRINCIPALES COMMANDES DE D-BUG 12

ASM - Commande de l'assembleur/désassembleur en ligne

Format de la ligne de commande : ASM <Ad>

Description des paramètres

<Ad>- Un nombre hexadécimal 16 bits ou une expression simple

Description de la commande

L'assembleur désassembleur est un éditeur mémoire interactif qui permet la visualisation et la modification du contenu mémoire sous forme de mnémoniques du langage assembleur. Chaque ligne source est traduite en code machine et rangée en mémoire dès qu'on la valide. A l'affichage du contenu mémoire, chaque instruction est désassemblée sous forme de mnémonique source et affichée avec le code machine hexa et les opérandes de l'instruction.

On peut écrire les mnémoniques et les opérandes indifféremment en majuscules ou minuscules, séparés par un ou plusieurs espaces. Par défaut, les valeurs numériques d'opérandes sont interprétées comme des nombres décimaux *signés*. L'écriture d'un $ devant un nombre les fait interpréter comme nombres hexa.

Après désassemblage et affichage d'une instruction, D-Bug12 affiche l'invite (>) à la suite. Si on tape alors un retour chariot, on désassemble l'instruction suivante et on l'affiche sur la ligne suivante. Si on tape une instruction CPU12 l'instruction est assemblée et rangée en mémoire. Le ligne contenant la nouvelle entrée est effacée et la nouvelle instruction est désassemblée et installée sur la même ligne, puis l'instruction suivante est désassemblée et affichée à l'écran.

Quand on entre des instructions de branchement, le nombre placé dans le champ opérande doit être l'adresse absolue de destination. L'assembleur calculera l'offset du branchement.

On termine la fonction d'assemblage/désassemblage par un point après l'invite de l'assembleur.

Exemple

```
asm 1000
xx:1000  CE1000        LDX   #$1000              >
xx:1003  09            DEX                       >
xx:1004  26FD          BNE   $1003               >
xx:1006  20FE          BRA   $1006               >.
>
```

BF - (Block Fill) Remplir un bloc mémoire d'une donnée

Format de ligne de commande : BF <Ad.début> <Ad.fin> [<Donnée>] [;nv]

Description des paramètres

< Ad.début > Un nombre hexa de 16 bits ou une expression simple.

< Ad.fin > Un nombre hexa de 16 bits ou une expression simple.

<Donnée> Un nombre hexa de 8-bits.

Description de la Commande

Remplit avec le même octet tous les emplacements d'une zone mémoire. < Ad.début > est le premier rempli et < Ad.fin > le dernier. Si on omet le paramètre <Donnée>, il a la valeur $00.

Exemple

```
>bf c00 fff 00
>bf x x+$ff 55
>
```

BR - (BReakpoint) Pose /Affiche les points d'arrêt

Format de ligne de commande : BR [<Ad.> …]

Description des paramètres

<Ad.> Un nombre hexa de 16 bits ou une expression simple.

Description de la Commande

Pose un point d'arrêt à une adresse donnée ou affiche les précédents. La fonction d'un point d'arrêt est d'arrêter l'exécution du programme quand il atteint l'adresse du point d'arrêt. Alors, D-Bug12 désassemble l'instruction au point d'arrêt, affiche le contenu des registres et attend une nouvelle commande D-Bug12 de l'utilisateur. On utilise ici l'option points d'arrêt logiciels, option activée par la commande « USEHBR OFF »

On pose les points d'arrêt en tapant la commande BR suivie d'une ou plusieurs adresses. Si on n'écrit pas d'adresse, on fait afficher les adresses des points d'arrêt actuels (10 au maximum).

Restrictions

D-Bug12 réalise la fonction de points d'arrêt logiciels en remplaçant le code de l'adresse choisie par l'instruction SWI. On ne peut pas mettre un point d'arrêt sur une instruction SWI utilisateur. On ne peut mettre un point d'arrêt que sur un code opératoire situé en RAM.

Exemple

```
>br 1100 1105
Breakpoints: 1100  1105
>br
Breakpoints: 1100  1105
>
```

BS - (Block Search), Cherche une adresse d'occurrence d'une séquence

Format de ligne de commande : BS <Ad.début> <Ad.fin> '<Ch>' | <Donnée> [<Donnée>]

Description des paramètres

<Ad.début> Un nombre hexa de 16 bits ou une expression simple.

<Ad.fin> Un nombre hexa de 16 bits ou une expression simple.

<Ch> Une chaîne ASCII composée de caractères quelconques sauf (').

<Donnée> Un nombre hexa de 8 bits ou une expression simple.

Description de la Commande

Cherche une séquence dans une zone d'adresses. On peut spécifier la séquence par une chaîne ASCII entre quotes ou par une suite de moins de neuf octets hexadécimaux. En cas de succès, l'adresse du premier octet s'affiche. Noter que si on définit la séquence par une chaîne ASCII, seuls sont autorisés les caractères imprimables à l'exception de la quote(').

Exemple

```
>bs c000 d000 'RTS'
Data Found at: $CF34
>bs 3000 4000 '12345'
Data Found at: $3D29
>bs f000 ffff 'bug'
Data Not Found
>
```

CALL - Exécute un sous-programme utilisateur

Format de la ligne de commande : CALL [<Ad.>]

Description du paramètre

<Ad.> Un nombre hexa de 16 bits ou une expression simple

Description de la commande

Exécute un sous-programme et retourne au moniteur D–Bug12 après exécution du RTS final.. Quand D–Bug12 reprend le contrôle, il affiche les contenus des registres du CPU au moment de l'exécution du RTS, à l'exception du compteur ordinal (PC) qui contient l'adresse de départ du sous-programme. Si on ne fournit pas d'adresse dans la ligne de commande, on utilise le contenu courant du PC comme adresse par défaut.

NOTE: La commande ne place pas de point d'arrêt avant de passer la main au code utilisateur.

Restrictions

Si le sous-programme appelé modifie la valeur du pointeur de pile, il DOIT rétablir le pointeur original avant d'exécuter son RTS final. Cette restriction est imposée du fait que D–Bug12 place 4 octets sur la pile utilisateur pour reprendre le contrôle après exécution du RTS final.

Exemple

```
>call 2000
Subroutine Call Returned

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 2000  3C00  1100  0000     01:00        1001 0000
xx:2000  42           INCA
>
```

G - (Go) Lancement de l'exécution du code utilisateur

Format de ligne de commande : G [<Ad.>]

Description des paramètres

<Ad.> Un nombre hexa de 16-bits ou une expression simple

Description de la commande

Lance l'exécution du code utilisateur en temps réel. Avant de commencer, tous les points d'arrêt posés par la commande BR sont placés en mémoire. L'exécution du programme utilisateur se poursuit jusqu'à rencontrer un point d'arrêt, une interruption ou l'appui sur le bouton reset du HC12EVB. Quand le code utilisateur s'arrête pour une de ces raisons et que D-Bug12 reprend le contrôle, un message s'affiche indiquant les raisons de l'arrêt. S'affichent également les contenus des registres du CPU12, et le désassemblage de l'instruction à l'adresse du PC courant. Si l'adresse de départ n'est pas précisée dans la ligne de commande, l'exécution du programme commence à l'adresse courante du PC.

Restrictions

Si le compteur ordinal pointe sur une instruction CALL, On ne peut pas utiliser la commande Go pour continuer l'exécution du programme. Cette limitation est due au fait que D-Bug12 trace une instruction avant de continuer l'exécution du programme. On ne peut pas tracer l'instruction CALL car cela interfère avec le fonctionnement de D-Bug12.

Exemple

```
>g 1000
User Bkpt Encountered

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 1004  3C00  0FFF  0000     01:00        1001 0000
xx:1004  26FD          BNE   $1003
>
```

GT - (Go unTil), Exécute le code utilisateur jusqu'à un point d'arrêt temporaire

Format de ligne de commande : GT <Ad.>

Description des paramètres

<Ad.> Un nombre hexa de 16-bits ou une expression simple

Description de la commande

Fonctionne comme Go sauf qu'elle place un point d'arrêt temporaire à l'adresse passée en ligne de commande. Les points d'arrêt définis par la commande BR ne sont PAS mis dans le code utilisateur avant le début de l'exécution du programme, qui commence à la valeur courante du PC. En arrivant au point d'arrêt, D-Bug12 reprend le contrôle en affichant un message expliquant la raison de l'arrêt du programme, les registres du CPU12, et l'instruction suivante.

Exemple

```
>gt 1006
Temporary Breakpoint Encountered

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 1006  3C00  0000  0000     01:00        1001 0100
xx:1006  20FE          BRA   $1006
```

LOAD - Charge un programme utilisateur au format S19

Format de ligne de commande : LOAD + F8

Description des paramètres : Voir notice complète.

Description de la commande

Charge des fichiers objets S19 en mémoire utilisateur à partir d'un dispositif extérieur. Les données ne s'affichent pas au fur et à mesure du chargement, mais toutes les dix lignes, une astérisque (*) s'affiche. Quand le chargement est réussi, D-Bug12 affiche son invite.

La commande Load est terminée quand D-Bug12 reçoit un enregistrement fin de fichier 'S8' ou 'S9'. Si le fichier objet ne contient pas d'enregistrement 'S8' ou 'S9', D–Bug12 n'envoie pas son invite et continue à attendre l'enregistrement de fin de fichier. Un appui sur le bouton de Reset ramènera D–Bug12 à son invite de ligne de commande.

Exemple

```
>load
**********
>
```

MD - (Memory Dump) Affiche la mémoire en octets hexa et en format ASCII

Format de ligne de commande : MD <Ad.départ> [<Ad.fin>]

Description des paramètres

<Ad.départ> Un nombre hexa de 16 bits ou une expression simple

<Ad.fin> Un nombre hexa de 16 bits ou une expression simple

Description de la commande

La commande affiche le contenu des mémoires à la fois sous forme d'octets en notation hexa et sous forme de caractères ASCII, par lignes de 16 octets.

Le paramètre <Ad.départ> est nécessaire, mais le paramètre <Ad.fin> est optionnel. S'il n'est pas fourni, une seule ligne s'affiche.

Le paramètre <Ad.départ> est arrondi au multiple de 16 inférieur, tandis que le paramètre <Ad.fin> est arrondi au multiple de 16 supérieur. Cela entraîne que chaque ligne affiche la mémoire depuis $xxx0 à $xxxF. Par exemple, si l'utilisateur a choisi $205 comme adresse de départ et $217 comme adresse de fin, la mémoire affichée va de $200 à $21F.

Exemple

```
>md 1000

1000  CE 10 00 09 - 26 FD 20 FE - 5A BD D6 C4 - 15 FB DE 76    ....&. .Z......v
>md f127 f145

F120  74 6C 6F 61 - 64 65 72 20 - 76 31 2E 30 - 2E 30 0D 0A    tloader v1.0.0..
F130  00 0D 0A 61 - 29 20 45 72 - 61 73 65 20 - 46 6C 61 73    ...a) Erase Flas
F140  68 0D 0A 62 - 29 20 50 72 - 6F 67 72 61 - 6D 20 46 6C    h..b) Program Fl
>
```

MDW - (Memory Dump Word) Affiche la mémoire en mots hexa et en format ASCII

Format de ligne de commande : MDW <Ad.départ> [<Ad.fin>]

Description des paramètres

<Ad.départ> Un nombre hexa de 16 bits ou une expression simple

<Ad.fin> Un nombre hexa de 16 bits ou une expression simple

Description de la commande

La commande affiche le contenu des mémoires à la fois sous forme de mots en notation hexa et sous forme de caractères ASCII, par lignes de 16 octets.

Le paramètre <Ad.départ> est nécessaire, mais le paramètre <Ad.fin> est optionnel. S'il n'est pas fourni, une seule ligne s'affiche.

Le paramètre <Ad.départ> est arrondi au multiple de 16 inférieur, tandis que le paramètre <Ad.fin> est arrondi au multiple de 16 supérieur -1. Cela entraîne que chaque ligne affiche la mémoire depuis $xxx0 à $xxxF. Par exemple, si l'utilisateur a choisi $205 comme adresse de départ et $217 comme adresse de fin, la mémoire affichée va de $200 à $21F.

Exemple

```
>mdw 1000

1000  CE10 0009 - 26FD 20FE - 5ABD D6C4 - 15FB DE76    ....&. .Z......v
>mdw f127 f145

F120  746C 6F61 - 6465 7220 - 7631 2E30 - 2E30 0D0A    tloader v1.0.0..
F130  000D 0A61 - 2920 4572 - 6173 6520 - 466C 6173    ...a) Erase Flas
F140  680D 0A62 - 2920 5072 - 6F67 7261 - 6D20 466C    h..b) Program Fl
>
```

MM - (Memory Modify) modifie les octets mémoire au format hexa

Format de ligne de commande : MM <Ad.> [<Donnée>]

Description des paramètres

< Ad. > Un nombre hexa de 16 bits ou une expression simple.

<Donnée> Un nombre hexa de 8 bits.

Description de la commande

Permet d'examiner et/ou de modifier le contenu mémoire sous forme de donnée hexa. Si le paramètre donnée 8 bits figure dans la ligne de commande, l'octet d'adresse < Adresse > est remplacé par <Donnée>. Sinon, D-Bug12 entre dans le mode interactif de modification mémoire. Dans ce mode, chaque octet est affiché sur une ligne séparée à la suite de son adresse. Une fois entrée la commande de modification mémoire, on dispose de plusieurs sous-commandes pour modifier et vérifier les contenus mémoire. Ces sous-commandes sont :

`[<Data>]<CR>` Mise à jour optionnelle de l'emplacement courant et affichage du suivant.

`[<Data>] / ou =` Mise à jour optionnelle de l'emplacement courant et réaffichage.

`[<Data>] ^ ou -` Mise à jour optionnelle de l'emplacement courant et affichage du précédent.

`[<Data>] .` Mise à jour optionnelle de l'emplacement courant et sortie de MM

Sauf le retour chariot, la sous-commande doit être séparée de la donnée entrée par au moins un espace. En cas de sous-commande invalide, un message d'erreur s'affiche et le contenu de l'emplacement courant est réaffiché.

Exemple

```
>mm 800
0800 00
0801 00 ff
0802 00 ^
0801 FF
0802 00
0803 00 55 /
0803 55 .
>
```

MMW - (Modify Memory Words) modifie les mots mémoire au format hexa

Format de ligne de commande : MMW <Ad.> [<Donnée>]

Description des paramètres

<Ad.> Un nombre hexa de 16 bits ou une expression simple.

<Donnée> Un nombre hexa de 16 bits.

Description de la commande

Permet d'examiner et/ou de modifier le contenu mémoire sous forme de donnée hexa 16 bits. Si le paramètre donnée 16 bits est présent dans la ligne de commande, l'octet d'adresse < Adresse > est remplacé par <Donnée>. Sinon, D-Bug12 entre dans le mode interactif de modification mémoire. Dans ce mode, chaque mot est affiché sur une ligne séparée à la suite de son adresse. Une fois entrée la commande de modification mémoire, on dispose de plusieurs sous-commandes pour modifier et vérifier les contenus mémoire. Ces sous-commandes sont :

`[<Data>]<CR>` Mise à jour optionnelle de l'emplacement courant et affichage du suivant.

`[<Data>] / or =` Mise à jour optionnelle de l'emplacement courant et réaffichage.

`[<Data>] ^ or -` Mise à jour optionnelle de l'emplacement courant et affichage du précédent.

`[<Data>] .` Mise à jour optionnelle de l'emplacement courant et sortie de MM

Sauf le retour chariot, la sous-commande doit être séparée de la donnée entrée par au moins un espace. En cas de sous-commande invalide, un message d'erreur s'affiche et le contenu de l'emplacement courant est réaffiché.

Exemple

```
>mmw 800
0800 00FF
0802 0055 aa55
0804 0000 ^
0802 AA55
0804 0000 843f
0806 0000 .
>
```

MOVE - Déplace un bloc de mémoire

Format de ligne de commande : MOVE <Ad.départ> <Ad.fin> <Ad.dest>

Description des paramètres

<Ad.départ> Un nombre hexa de 16 bits ou une expression simple.

<Ad.fin> Un nombre hexa de 16 bits ou une expression simple.

<Ad.dest> Un nombre hexa de 16 bits ou une expression simple.

Description de la commande

La commande sert à déplacer un bloc de mémoire d'un emplacement à un autre octet par octet. Le nombre d'octets déplacés est un de plus que <Ad.fin> - <Ad.départ>. Le bloc de mémoire créé commence à l'adresse <Ad.dest>. Il peut recouvrir partiellement le bloc de départ.

Restrictions

On peut déplacer au minimum 1 octet si <Ad.départ> est égal à <Ad.fin>. Au maximum, on peut en déplacer 2^{16} - 1.

Exemple

```
>move 800 8ff 1000
>
```

NOBR - (NO BReakpoint) Supprime un/tous les point(s) d'arrêt.

Format de ligne de commande : NOBR [<Ad.> …]

Description des paramètres

<Ad.> Un nombre hexa de 16 bits ou une expression simple

Description de la commande

Supprime un ou plusieurs des points d'arrêt précédemment entrés. En l'absence d'argument, tous les points d'arrêt sont supprimés de la table.

Exemple

```
>br 800 810 820 830
Breakpoints: 0800 0810 0820 0830
>nobr 810 820
Breakpoints: 0800 0830
>nobr
All Breakpoints Removed
>
```

RD - (Registers Display) Affiche les contenus des registres du CPU12

Format de ligne de commande : RD

Description de la commande

La commande sert à afficher les registres du CPU12. Les registres s'affichent dans le même format que lorsqu'on rencontre un point d'arrêt.

Exemple

```
>rd

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 1006  3C00  0000  0000     01:00        1001 0100
xx:1006  20FE          BRA   $1006
>
```

RM - (Register Modify) Modification interactive du contenu des registres du CPU12

Format de ligne de commande : RM

Description de la commande

Examine et/ou modifie les contenus des registres du CPU12 de manière interactive. Lors de l'affichage de chaque registre, D-Bug12 permet à l'utilisateur d'entrer une nouvelle valeur sous forme hexa. Si on ne désire pas modifier, un retour chariot passe au registre suivant. Après affichage du dernier registre (CCR) la commande boucle sur le premier registre (PC). On sort du mode interactif en tapant un point (.) en premier caractère.

Les registres s'affichent dans l'ordre suivant : PC, SP, X, Y, A, B, CCR.

Exemple

```
>rm

PC=1006 1200
SP=3C00
IX=0000 1000
IY=0000
A=01
B=00
CCR=94 90
PC=1200 .
>
```

SO - (Step Over) Saut d'un appel de sous-programme

Format de ligne de commande : SO

Description de la commande

Quand on trace du code il n'est souvent pas nécessaire de tracer dans les sous-programmes déjà corrects. La commande SO est semblable à la commande trace (T), sauf que les appels de sous-programme (BSR, JSR and CALL) sont tracés comme une seule instruction. Quand la commande SO rencontre une de ces instructions, elle place un point d'arrêt temporaire sur l'instruction suivant le BSR, JSR ou CALL. Elle lance ensuite une commande Go, exécutant le sous-programme à pleine vitesse. Les autres instructions sont exécutées comme en Trace.

Exemple

```
>so

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 1001  3C00  1000  0000     01:01        1001 0000
xx:1001  162000        JSR   $2000
>so

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 1004  3C00  1000  0000     02:01        1001 0000
xx:1004  52            INCB

>
```

T - (Trace) Exécute pas à pas les instructions CPU12

Format de ligne de commande : T [<Count>]

Description des paramètres

<Count> Un nombre décimal de 1 à 255

Description de la commande

On utilise la commande Trace pour exécuter une ou plusieurs instructions d'un programme utilisateur à partir de la valeur courante du compteur ordinal (PC). A chaque instruction, les contenus des registres du CPU12 s'affichent ainsi que la *prochaine* instruction à exécuter. Pour exécuter une seule instruction, entrer la commande trace suivie d'un retour chariot.

Restrictions

Il n'est pas possible de tracer les instructions qui bouclent sur elles mêmes (`BCC`, `LBCC`, `BRSET`,`BRCLR`,`BRSET`,`BRCLR`,`DBEQ/NE`,`IBEQ/NE`,`TBEQ/NE`) à cause de la méthode utilisée pour exécuter une seule instruction.On peut contournetr la difficulté en utilisant la gommande GT (Go Till) qui pose un point d'arrêt temporaire sur l'instruction suivante.

L'instruction CALL ne peut être tracée car elle interfère avec le fonctionnement de D-Bug12.

Exemple

```
>t

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 1001  3BFE  1000  0000     00:DF        1001 0100
xx:1001  162000        JSR   $2000

>t 2

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 2000  3BFC  1000  0000     00:DF        1001 0100
xx:2000  42            INCA

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 2001  3BFC  1000  0000     01:DF        1001 0000
xx:2001  3D            RTS

>
```

<Nom de registre> - Modifie la valeur d'un registre CPU12

Format de ligne de commande

< Nom de registre > <Valeur>

Description des paramètres <Nom de registre> est un des noms de registre CPU12 suivants :

Nom	Description	Valeurs
PC	Compteur ordinal	$0..$FFFF
SP	Pointeur de pile	$0..$FFFF
X	Registre d'index X	$0..$FFFF
Y	Registre d'index Y	$0..$FFFF

A	Accumulateur A	$0..$FF
B	Accumulateur B	$0..$FF
D	Accumulateur D(A:B)	$0..$FFFF
CCR	Registre Code Condition	$0..$FF
PP	Registre PPAGE	$0..$FF

Chaque champ du CCR peut être modifié sous les noms suivants :

CCR	Nom du Bit	Valeur
S	Validation du STOP	0/1
H	Demi Carry	0/1
N	Indicateur Négatif	0/1
Z	Indicateur Zero	0/1

CCR	Nom du Bit	Valeur
V	Indicateur Overflow	0/1
C	Indicateur Carry	0/1
IM	Masque d'IRQ	0/1
XM	Masque d'XIRQ	0/1

Pour chaque registre, <Valeur> peut être un nombre hexadécimal ou une expression simple.

Description de la commande

Cet ensemble de « commandes » utilise les noms des registres du CPU comme commandes individuelles pour permettre de changer leur contenu.

Exemple

```
>pc 700e

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 700E  3C00  1000  0000     02:01        1001 0000
xx:700E  FFFFFF        LDS   $FFFF
>x 1000

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 700E  3C00  1000  0000     02:01        1001 0000
xx:700E  FFFFFF        LDS   $FFFF
>c 1

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 700E  3C00  1000  0000     02:01        1001 0001
xx:700E  FFFFFF        LDS   $FFFF
>d adf

PP  PC    SP    X     Y    D = A:B   CCR = SXHI NZVC
38 700E  3C00  1000  0000     0A:DF        1001 0001
xx:700E  FFFFFF        LDS   $FFFF
>
```

ROUTINES UTILISATEUR

Conventions d'appel

Tous les sous-programmes utilisateurs ont été écrits en C. En général, les paramètres sont passés par la pile. Il faut pousser les paramètres sur la pile dans l'ordre inverse où ils sont déclarés dans la fonction (de droite à gauche) *excepté* le dernier paramètre (le premier paramètre de la déclaration de la fonction C) qui est passé par l'accumulateur D.

Les fonctions n'ayant qu'un paramètre le passent toujours par D. Les paramètres `char` doivent toujours être convertis `en int`. Cela signifie que même si un paramètre est déclaré comme `char` il occupera deux octets de pile. Les paramètres `char` doivent occuper l'octet poids faible (octet d'adresse haute) d'un mot poussé dans la pile, ou l'accumulateur B si le paramètre est passé par D.

Les paramètres poussés en pile avant l'appel de la fonction y restent au retour de la fonction. C'est la responsabilité du programme *appelant* de les retirer de la pile.
Tous les résultats à 8 ou 16 bits sont retournés dans l'accumulateur D. Les valeurs `char` retournées sont situées dans l'accumulateur B. Les résultats `Boolean` des fonctions sont zéro pour Faux et non zéro pour Vrai.

A l'exception du pointeur de pile, aucun registre du CPU12 n'est préservé par les fonctions appelées. S'il faut préserver certaines valeurs de registres, on doit les pousser en pile avant les paramètres et les récupérer après avoir désalloué les paramètres.
Sur la figure C-1, les fonctions avec le qualificateur `far` sont implantées en mémoire paginée du MC9S12DP256 et on doit les appeler à l'aide de l'instruction CALL Noter que leurs pointeurs occupent quatre octets au lieu de deux.

Interface avec le langage d'assemblage

L'appel de fonctions à partir du langage assembleur est un simple problème d'empilement des paramètres dans le bon ordre, en chargeant le premier ou le seul paramètre dans l'accumulateur D. On peut alors appeler la fonction par une instruction CALL ou JSR. Le code qui suit l'appel doit retirer tous les paramètres de la pile. Si on n'a mis en pile un seul paramètre, le moyen le plus efficace de le retirer est une simple instruction PULX ou PULY. S'il y en a davantage, il vaut mieux utiliser l'instruction LEAS.
Si on a sauvé des registres du CPU12 sur la pile avant de passer les paramètres, il faut les récupérer avec les instructions PUL correspondantes. La figure C-2 donne un exemple d'appel de la fonction `WriteEEByte()`:

```
WriteEEByte:equ $EEA6   ; adresse du pointeur de fonction.
;
            ldab  #$55  ; écrire $55 dans l'EEPROM.
            pshd        ; place la donnée sur la pile.
            ldd   EEAd  ; adresse d'écriture.
            call  [WriteEEByte,pcr] ;Appel du sous-programme.
            pulx        ; suppression du paramètre de la pile.
            beq   Error ; Un retour 0 signifie erreur.
```

Figure C-2, Appel du sous-programme `WriteEEByte`

Function	Description	Pointer Address
far main()	Début de D-Bug12	$EE80
getchar()	Récupère un caractère de SCI0 ou SCI1	$EE84
putchar()	Envoie un caractère sur SCI0 ou SCI1	$EE86
printf()	Sortie formatée - Traduit les valeurs binaires en caractères	$EE88
far GetCmdLine()	Obtain a line of input from the user	$EE8A
far sscanhex()	Convertit une chaîne hexadécimale ASCII en entier binaire	$EE8E
isxdigit()	Teste l'appartenance à l'ensemble [0..9, a..f, A..F]	$EE92
toupper()	Convertit minuscules en majuscules	$EE94
isalpha()	Teste l'appartenance à l'ensemble[a..z, A..Z]	$EE96
strlen()	Retourne la longueur d'une chaîne terminée par un octet nul	$EE98
strcpy()	Copie une chaîne terminée par un octet nul	$EE9A
far out2hex()	Affiche un octet sous forme de 2 caractères hexadécimaux	$EE9C
far out4hex()	Affiche un mot de 16 bits sous forme de 4 caractères hex	$EEA0
SetUserVector()	Setup user interrupt service routine	$EEA4
far WriteEEByte()	Ecrit un octet de donnée sur l'EEPROM interne	$EEA6
far EraseEE()	Efface toute l'EEPROM interne	$EEAA
far ReadMem()	Lit des données dans la carte mémoire du M68HC12	$EEAE
far WriteMem()	Ecrit des données sur la carte mémoire du M68HC12	$EEB2

Figure C-1, Liste des fonctions utilisateur

`void far main(void);` Adresse du pointeur: $EE80
Point d'entrée « tiède » de D-Bug12. Parmi de nombreuses utilisations (voir notice complète), permet un retour au moniteur en fin de programme utilisateur. **Attention** : un tel retour depuis un programme utilisateur par la fonction `main()` réinitialise toutes les tables internes et les variables de D-Bug12. Tout point d'arrêt est perdu et les SWI des points d'arrêt restent dans le programme utilisateur.

`int getchar(void);` Adresse du pointeur: $EE84
La fonction `getchar()` permet de récupérer un caractère depuis le SCI du terminal de contrôle. S'il n'y a pas de caractère dans le registre de réception du SCI à l'appel de la fonction, `getchar()` attendra la réception d'un caractère. Comme la fonction retourne un entier (`int`), le caractère 8 bits est placé dans l'accumulateur B.

`int putchar(int);` Adresse du pointeur: $EE86
La fonction `putchar()` permet d'envoyer un caractère sur le SCI du terminal de contrôle. Si le registre de transmission est plein à l'appel de la fonction, `putchar()` attendra qu'il soit vide pour envoyer le caractère. Il n'y a pas de buffering du caractère. `putchar()` retourne le caractère envoyé. Cependant, il ne détecte aucune condition d'erreur et ne retournera donc jamais EOF.Comme le caractère retourné est un `int`, il est placé dans l'accumulateur B.

`int printf(char *format, ...);` Adresse du pointeur: $EE88
La fonction `printf()` sert à convertir, formater et imprimer ses arguments sur la sortie standard sous le contrôle de la chaîne de format pointée par `format`. Elle retourne le nombre de caractères envoyés vers la sortie standard. La version de `printf()` du moniteur supporte l'écriture formatée de tous les types de données *exceptés* les formats flottants. La chaîne de format peut contenir deux types d'objets de base : les caractères ASCII qui sont copiés

directement de la chaîne format vers la sortie standard, et les spécifications de conversion qui permettent de convertir, de formater et d'envoyer les arguments successifs de `printf()`. Chaque spécification de conversion commence par un signe pourcent (%) et se termine par un seul caractère de conversion. Les caractères optionnels de format peuvent apparaître entre le signe pourcent et le caractère de conversion dans l'ordre suivant :
[-][<Largeur de champ>][.][<Précision>][h | l]

Caractères	**Description**
- (signe moins)	Justification à gauche de l'argument converti.
Largeur de champ	Entier qui précise la largeur minimum du résultat. L'argument sera affiché sur au moins cette largeur. Il sera au besoin complété à droite ou à gauche.
. (point)	Sépare la largeur de champ de la précision.
Précision	Entier qui précise le nombre maximum de caractères à afficher pour une chaîne ou le nombre minimum de chiffres pour un entier.
h	Pour afficher un entier comme un short.
l (lettre)	Pour afficher un entier comme un long

Les champs Largeur ou précision peuvent contenir une astérisque(*) au lieu d'un nombre. The asterisk will cause the value of next argument in the argument list to be used instead.
La figure C-3 ci dessous contient les caractères de conversion pris en compte par la fonction `printf()` de D-Bug12. Si le(s) caractère(s) de conversion qui sui(ven)t le caractère pourcent n'est ni un caractère de format ni un de ceux là, le comportement de la fonction `printf()` est indéfini.

Caractère	Type; Affiché comme
d, i	int; nombre décimal signé
o	int; nombre octal non signé (sans 0 initial)
x	int; nombre hexa non signé utilisant abcdef pour 10..15
X	int; nombre hexa non signé utilisant ABCDEF pour 10..15
u	int; nombre décimal non signé
c	int; simple caractère
s	char *; affiche une chaîne jusqu'à un '\0'
p	void *; pointer (représentation dépendant de l'implémentation)
%	pas d'argument converti, imprime un a %

Figure C-3, Caractères de conversion de printf()

Pour ceux qui sont familiers avec le C ou la fonction `printf()` les exemples suivants montrent les résultats de la fonction `printf()` pour différentes chaînes de formats.
Exemple 1.

```
printf("Signed Decimal: %d Unsigned Decimal: %u/n", Num, Num);
```

Où Num a pour valeur $FFFF, affiche le résultat :

```
Signed Decimal: -1 Unsigned Decimal: 65535
```

Exemple 2.

```
printf("Hexadecimal: %H Hexadecimal: %4.4H/n", Num, Num);
```

Où Num a la valeur $FF, affiche le résultat :

```
Hexadecimal: FF Hexadecimal: 00FF
```

Exemple 3.

```
printf("This is a %s/n", TestStr);
```

Où TestStr est un *pointeur* sur (adresse de) le premier octet d'un tableau de caractères terminé par un NULL contenant "Test".affiche le résultat :

```
This is a Test
```

`int far GetCmdLine(char *CmdLineStr, int CmdLineLen);`
Adresse du pointeur: $EE8A
La fonction `GetCmdLine()` permet de récupérer une ligne entrée par l'utilisateur. `GetCmdLine()` reçoit un caractère à la fois par l'intermédiaire de getchar(). Le caractère est renvoyé en écho par putchar() et rangé dans le tableau de caractères pointé par `CmdLineStr`. On peut entrer au maximum `CmdLineLen - 1` caractères ASCII imprimables. Les seuls caractères non imprimables traités sont le caractère backspace ($08) et le caractère retour chariot ($0D). Tout autre caractère non imprimable est ignoré.
Le caractère backspace ($08) est utilisé par la fonction `GetCmdLine()` pour supprimer du buffer de ligne de commande le dernier caractère reçu. Quand `GetCmdLine()` reçoit le caractère backspace, elle renvoie le backspace en écho au terminal, imprime le caractère ASCII espace ($20), et envoie un second caractère backspace au terminal. Cette action a pour effet d'effacer le dernier caractère à l'écran. En même temps, il est supprimé du buffer de ligne de commande. Un backspace reçu en premier caractère est ignoré.
Un retour chariot ($0D) termine la réception de caractères. Cependant, le retour chariot n'est pas placé dans le buffer de commande. On y met à la place le caractère NULL ($00).
Avant de finir, tous les caractères entrés sont convertis en majuscules. `GetCmdLine()` renvoie toujours un code de `noErr`.

`char * far sscanhex(char *HexStr, unsigned int *BinNum);`
Adresse du pointeur : $EE8E
La fonction `sscanhex()` sert à convertir une chaîne hexadécimale ASCII en entier binaire. La chaîne pointée par `HexStr` peut contenir un nombre quelconque de caractères hexa. Cependant, la valeur convertie ne doit pas être supérieure à $FFFF. La chaîne doit se terminer soit par un espace ($20) soit par un NULL ($00). `sscanhex()` retourne soit un pointeur sur le caractère terminal soit un pointeur NULL. Un pointeur NULL indique soit la présence d'un caractère non hexa dans la chaîne, soit une valeur supérieure à $FFFF après conversion.

`int isxdigit(int c);` Adresse du pointeur: $EE92
La fonction `isxdigit()` teste l'appartenance du caractère c à l'ensemble [0..9, a..f, A..F]. Si le caractère appartient à l'ensemble, la fonction retourne une valeur non nulle (vrai), sinon elle retourne une valeur nulle (faux).

`int toupper(int c);` Adresse du pointeur: $EE94
Si `c` est une minuscule, `toupper()` retourne la majuscule correspondante. Si c est majuscule, retourne simplement c.

`int isalpha(int c);` Adresse du pointeur: $EE96
La fonction `isalpha()` teste l'appartenance du caractère c à l'ensemble [a..z, A..Z]. Si le caractère appartient à l'ensemble, la fonction retourne une valeur non nulle (vrai), sinon elle retourne une valeur nulle (faux).

`unsigned int strlen(const char *cs);` Adresse du pointeur: $EE98
La fonction `strlen()` retourne la longueur de la chaîne pointée par `cs`. Une chaîne est un tableau de caractères terminée par un caractère `'\0'`.

`char * strcpy(char *s1, char *s2);` Adresse du pointeur: $EE9A
La fonction `strcpy()` copie le contenu de la chaîne `s2` dans la chaîne pointée par `s1` y compris le `'\0'`. Retourne un pointeur sur `s1`.

`void far out2hex(unsigned int num);` Adresse du pointeur: $EE9C
La fonction `out2hex()` affiche l'octet poids faible de `num` sur le terminal de contrôle sous forme de deux caractères hexa. L'octet poids fort est ignoré. On fournit cette fonction pour ceux qui ne maîtrisent pas bien la fonction `printf()`. `out2hex()` appelle simplement printf() avec la chaîne de format `"%2.2X"`.

`void far out4hex(unsigned int num);` Adresse du pointeur: $EEA0
`out4hex()` affiche `num` sur le terminal de contrôle sous forme de quatre caractères hexa.
On fournit cette fonction pour ceux qui ne maîtrisent pas bien la fonction `printf()`. `out4hex()` appelle simplement printf() avec la chaîne de format `"%4.4X"`.

`int SetUserVector (int VectNum,Address UserAddress);`
Adresse du pointeur: $EEA4
Non documenté. Voir notice générale.

`Boolean far WriteEEByte(Address EEAddress, Byte EEData);`
Adresse du pointeur: $EEA6
`WriteEEByte()` fournit un mécanisme pour programmer individuellement des octets de l'EEPROM interne sans avoir à manipuler les registres de contrôle de programmation de l'EEPROM.. `WriteEEByte()` ne fait aucun test sur `EEAddress` pour s'assurer qu'il retombe bien dans la zone d'adresse de l'EEPROM interne. Un programme utilisateur peut déterminer l'adresse de départ et la taille de la mémoire EEPROM interne en examinant le contenu des champs `CustData.EEBase` et `CustData.EESize`. La fonction effectue un effacement d'octet avant de programmer, puis vérifie le résultat de l'opération. En cas d'échec, la fonction retourne une valeur faux (zéro).

`int far EraseEE(void);` Adresse du pointeur: $EEAA
La fonction `EraseEE()` fournit un mécanisme d'effacement global de l'EEPROM interne sans avoir à manipuler les registres de contrôle de programmation de l'EEPROM. Une fois terminée l'opération d'effacement, on teste toute la mémoire définie par `CustData.EEBase` et `CustData.EESize`. Si tous les octets ne sont pas à `0xff`, retour d'un code d'erreur non-nul.

`int far ReadMem (Address StartAddress, Byte *MemDataP,`
`unsigned int NumBytes);` Adresse du pointeur: $EEAE
La fonction `ReadMem()`est utilisée de façon interne par D-Bug12 pour tous ses accès en lecture mémoire. Pour l'implémentation du moniteur, la fonction `ReadMem()` lit simplement `NumBytes` de données directement sur la mémoire cible et les range dans un buffer pointé par `MemDataP`. Pour un utilisateur, cette fonction n'a probablement pas d'intérêt. Mieux vaut une lecture directe. Retour d'un code d'erreur non nul en cas de problème.

`int WriteMem (Address StartAddress, Byte *MemDataP,`
`unsigned int NumBytes);` Adresse du pointeur: $EEB2
La fonction `WriteMem()` est utilisée de façon interne par D-Bug12 pour tous ses accès en écriture mémoire. `WriteMem()`. Elle diffère de `ReadMem()` du fait qu'elle traite l'EEPROM interne. Si l'octet est dans la zone définie par `CustData.EEBase` et `CustData.EESize`, `WriteMem()`fait appel à la fonction `WriteEEByte()` pour programmer la donnée dans la mémoire EEPROM interne. Retour d'un code d'erreur non nul en cas de problème.

RELAIS RAM DES VECTEURS D'INTERRUPTIONS

D-Bug12 comporte des gestionnaires d'interruption par défaut pour tous les vecteurs d'interruptions reconnus par MC9S12DP256. Cependant, pour permettre au programmeur de travailler sous interruption, D-Bug12 fournit une table de vecteurs en RAM. Chacune des 64 entrées de la table consiste en une adresse sur deux octets, la table commençant en $3E00. Initialement, toutes les cases du tableau contiennent l'adresse $0000. Y mettre une adresse différente entraîne l'exécution de la tâche pointée par l'adresse quand l'interruption associée se produit. La tâche fournie par l'utilisateur doit se terminer par l'instruction RTI pour que le programme principal puisse reprendre là où il avait été interrompu.
En cas d'interruption non masquable, si la case correspondante du tableau contient l'adresse par défaut $0000, D-Bug12 prend la main et affiche un message indiquant la source de l'interruption ainsi que les registres du CPU au moment de l'interruption..
La Figure 4 montre la correspondance entre la source d'interruption et le vecteur en RAM sur deux octets. Noter que bien qu'il y ait une case dans la table pour SCI0, un remplacement de cette adresse sera ignoré puisque D-Bug12 se sert du SCI0 pour toute sa communication.

Interrupt Source	RAM Vector
Réservé $FF80	$3E00
Réservé $FF82	$3E02
Réservé $FF84	$3E04
Réservé $FF86	$3E06
Réservé $FF88	$3E08
Réservé $FF8A	$3E0A
Arrêt d'urgence PWM	$3E0C
Interruption Port P	$3E0E
MSCAN 4 Transmit	$3E10
MSCAN 4 Receive	$3E12
MSCAN 4 Errors	$3E14
MSCAN 4 Wake-up	$3E16
MSCAN 3 Transmit	$3E18
MSCAN 3 Receive	$3E1A
MSCAN 3 Errors	$3E1C
MSCAN 3 Wake-up	$3E1E
MSCAN 2 Transmit	$3E20
MSCAN 2 Receive	$3E22
MSCAN 2 Errors	$3E24
MSCAN 2 Wake-up	$3E26
MSCAN 1 Transmit	$3E28
MSCAN 1 Receive	$3E2A
MSCAN 1 Errors	$3E2C
MSCAN 1 Wake-up	$3E2E
MSCAN 0 Transmit	$3E30
MSCAN 0 Receive	$3E32
MSCAN 0 Errors	$3E34
MSCAN 0 Wake-up	$3E36
Flash	$3E38
EEPROM	$3E3A
SPI2	$3E3C
SPI1	$3E3E

Interrupt Source	RAM Vector
Bus IIC	$3E40
DLC	$3E42
SCME	$3E44
CRG Lock	$3E46
Débordement Pulse Accumulator B	$3E48
Modulus Down Counter Underflow	$3E4A
Interruption Port H	$3E4C
Interruption Port J	$3E4E
ATD1	$3E50
ATD0	$3E52
SCI1	$3E54
SCI0	$3E56
SPI0	$3E58
Pulse Accumulator A Input Edge	$3E5A
Débordement Pulse Accumulator A	$3E5C
Débordement Timer	$3E5E
Timer Canal 7	$3E60
Timer Canal 6	$3E62
Timer Canal 5	$3E64
Timer Canal 4	$3E66
Timer Canal 3	$3E68
Timer Canal 2	$3E6A
Timer Canal 1	$3E6C
Timer Canal 0	$3E6E
Interruption temps réel	$3E70
IRQ	$3E72
XIRQ	$3E74
SWI	$3E76
Instruction Non implementée Trap	$3E78
N/A	$3E7A
N/A	$3E7C
N/A	$3E7E

Figure 4 Adresses des vecteurs d'interruption en RAM

CARTE MEMOIRE DE D-BUG12 EN MODE EVB

Quand D-Bug12 est configuré en mode EVB, il fonctionne comme un moniteur résident en ROM. Comme tous les moniteurs résidents, il utilise des ressources du composant pour ses fonctions de mise au point. La table de la figure 5 ci-dessous détaille la carte mémoire de D-Bug12 opérant sur une version à 256K. D-Bug12 utilise 1024 octets de la RAM interne du MCU. Les 11K de RAM restants sont disponibles pour le développement de programmes d'application. Noter que D-Bug12 en mode EVB ne permet pas d'utiliser la mémoire Flash du MCU.

Zone d'adresses	**Description**
$0000 - $03FF	Registres Micro-contrôleur
$0400 - $0FFF	EEPROM interne
$1000 - $3BFF	RAM interne (utilisateur)
$3C00 - $3FFF	RAM interne (D-Bug12)
$4000 - $EE7F	Code D-Bug12
$EE80 - $EEBF	Table des routines utilisateur
$EEC0 - $EEFF	Table des données Customization
$EF00 - $EF8B	Code de démarrage de D-Bug12
$EF8C - $EFFF	Table secondaire de Reset/Inter.
$F000 - $FFFF	Chargeur bootstrap

Figure 5 - Carte mémoire de D-Bug12 en mode EVB pour le MC9S12Dx256

Noter que même si le MC9S12Dx256 contient 4K octets d'EEPROM, seuls sont accessibles les 3Koctets d'en haut puisque le Koctet inférieur est recouvert par le bloc des registres.
De même, bien que la table ci-dessus décrive une carte mémoire de 64K, le code de D-Bug12 occupe de la mémoire Flash paginée commençant à la page $38. En outre, la plupart des pages Fash fixes sont utilisées par D-Bug12.

Adresses d'interruption

en mémoire Flash et en RAM (sous D-Bug 12)

Source d'interruption	Adresse Flash	Adresse RAM
Reserved	$FF80	$3E00
Reserved	$FF82	$3E02
Reserved	$FF84	$3E04
Reserved	$FF86	$3E06
Reserved	$FF88	$3E08
Reserved	$FF8A	$3E0A
PWM Emergency Shutdown	$FF8C	$3E0C
Port P Interrupt	$FF8E	$3E0E
MSCAN 4 Transmit	$FF90	$3E10
MSCAN 4 Receive	$FF92	$3E12
MSCAN 4 Errors	$FF94	$3E14
MSCAN 4 Wake-up	$FF96	$3E16
MSCAN 3 Transmit	$FF98	$3E18
MSCAN 3 Receive	$FF9A	$3E1A
MSCAN 3 Errors	$FF9C	$3E1C
MSCAN 3 Wake-up	$FF9E	$3E1E
MSCAN 2 Transmit	$FFA0	$3E20
MSCAN 2 Receive	$FFA2	$3E22
MSCAN 2 Errors	$FFA4	$3E24
MSCAN 2 Wake-up	$FFA6	$3E26
MSCAN 1 Transmit	$FFA8	$3E28
MSCAN 1 Receive	$FFAA	$3E2A
MSCAN 1 Errors	$FFAC	$3E2C
MSCAN 1 Wake-up	$FFAE	$3E2E
MSCAN 0 Transmit	$FFB0	$3E30
MSCAN 0 Receive	$FFB2	$3E32
MSCAN 0 Errors	$FFB4	$3E34
MSCAN 0 Wake-up	$FFB6	$3E36
Flash	$FFB8	$3E38
EEPROM	$FFBA	$3E3A
SPI2	$FFBC	$3E3C
SPI1	$FFBE	$3E3E

Source d'interruption	Adresse Flash	Adresse RAM
IIC Bus	$FFC0	$3E40
DLC	$FFC2	$3E42
SCME	$FFC4	$3E44
CRG Lock	$FFC6	$3E46
Pulse Accumulator B Overflow	$FFC8	$3E48
Modulus Down Counter Underflow	$FFCA	$3E4A
Port H Interrupt	$FFCC	$3E4C
Port J Interrupt	$FFCE	$3E4E
ATD1	$FFD0	$3E50
ATD0	$FFD2	$3E52
SCI1	$FFD4	$3E54
SCI0	$FFD6	$3E56
SPI0	$FFD8	$3E58
Pulse Accumulator A Input Edge	$FFDA	$3E5A
Pulse Accumulator A Overflow	$FFDC	$3E5C
Timer Overflow	$FFDE	$3E5E
Timer Channel 7	$FFE0	$3E60
Timer Channel 6	$FFE2	$3E62
Timer Channel 5	$FFE4	$3E64
Timer Channel 4	$FFE6	$3E66
Timer Channel 3	$FFE8	$3E68
Timer Channel 2	$FFEA	$3E6A
Timer Channel 1	$FFEC	$3E6C
Timer Channel 0	$FFEE	$3E6E
Real Time Interrupt	$FFF0	$3E70
IRQ	$FFF2	$3E72
XIRQ	$FFF4	$3E74
SWI	$FFF6	$3E76
Unimplemented Instruction Trap	$FFF8	$3E78
N/A	$FFFA	$3E7A
N/A	$FFFC	$3E7C
N/A	$FFFE	$3E7E

Registres du micro-contrôleur 9S12DG256B

Le micro-contrôleur 9S12DG256B a d'énormes possibilités, ce qui se traduit par un grand nombre de registres, au mode d'emploi parfois compliqué. Les pages qui suivent fournissent un mode d'emploi résumé des proncipaux registres. Si ce résumé n'est pas suffisant, il faut se reporter aux documents originaux qui sont disponibles sur le réseau dans les fichiers suivants :

DISTRI\DOCHCS12\ATD.PDF
DISTRI\DOCHCS12\BDLC.PDF
DISTRI\DOCHCS12\COEUR.PDF
DISTRI\DOCHCS12\CRG.PDF
DISTRI\DOCHCS12\ECT.PDF
DISTRI\DOCHCS12\EEPROM.PDF
DISTRI\DOCHCS12\FLASH.PDF
DISTRI\DOCHCS12\IIC.PDF
DISTRI\DOCHCS12\MICRO.PDF
DISTRI\DOCHCS12\MSCAN.PDF
DISTRI\DOCHCS12\PIM.PDF
DISTRI\DOCHCS12\PWM.PDF
DISTRI\DOCHCS12\SCI.PDF
DISTRI\DOCHCS12\SPI.PDF
DISTRI\DOCHCS12\VREG.PDF

Ces fichiers définissent les adresses relatives des registres, en prenant comme adresse 0 celle du premier registre du module. On trouvera les adresses réelles d'implantation des modules dans le fichier

DISTRI\DOCHCS12\9S12DP256B.PDF

REGISTRES DU COEUR (MEBI + MMC + INT +BKP + CRG + Divers)

Registre de données 8 bits du Port A (MEBI)

$0000	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	**PORTA**
Reset	-	-	-	-	-	-	-	-	

en mode étendu, bus d'adresses (A15-A8) et de données (D15 - D8)

Registre de données 8 bits du Port B (MEBI)

$0001	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	**PORTB**
Reset	-	-	-	-	-	-	-	-	

en mode étendu, bus d'adresses (A7-A0) et de données (D7 - D0)

Registre de direction de données 8 bits du Port A (MEBI)

$0002	DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0	**PORTB**
Reset	0	0	0	0	0	0	0	0	

DDRAx à 1 / configure PAx en sortie, à 0 / configure PAx en entrée

Registre de direction de données 8 bits du Port B (MEBI)

$0003	DDRB7	DDRB6	DDRB 5	DDRB 4	DDRB 3	DDRB 2	DDRB 1	DDRB 0	**DDRB**
Reset	-	-	-	-	-	-	-	-	

DDRBx à 1 / configure PBx en sortie, à 0 / configure PBx en entrée

Registres de $0004 à $0007 réservés

Registre de données 8 bits du Port E (MEBI)

$0008	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0	**PORTE**
Reset	-	-	-	-	-	-	-	-	
Autres	NOACC	MODB	MODA	ECLK	LSTRB	R/W	IRQ	XIRQ	

En mode étendu, contient les contrôles des échanges sur bus, programmés par PEAR.

Registre de direction de données 8 bits du Port E (MEBI)

$0009	DDRE7	DDRE6	DDRE5	DDRE4	DDRE3	DDRE2	0	0	**DDRE**
Reset	0	0	0	0	0	0	0	0	

DDREx à 1 / configure PEx en sortie, à 0 / configure PEx en entrée. Les lignes PE0 et PE1 sont toujours en entrée

Registre d'assignement du Port E (MEBI)

$000A	NOACCE	0	PIPOE	NECLK	LSTRE	RDWE	0	0	**PEAR**
Reset	0	0	0	0	1	0	0	0	

En mode single chip et sur la carte d'application, le port E garde sa fonction de port

Registre de mode (MEBI)

$000B	MODC	MODB	MODA	0	IVIS	0	EMK	EME	**MODE**
Reset	1	0	0	0	0	0	0	0	

Avec la carte T-Board, on est en mode normal single chip (MODC = 1, MODA = MODB = 0)
Sur la carte, on lit $80, c'est à dire que IVIS = EMK = EME = 0

Registre de contrôle de pullup des porte A, B, E, K (MEBI)

$000C	PUPKE	0	0	PUPEE	0	0	PUPBE	PUPAE	**PUCR**
Reset	1	0	0	1	0	0	0	0	

1 / Toutes les lignes du port sont munies de pullups. 0 / Pas de pullup sur les entrées du port. Sur la carte, on lit $90

Registre de réduction desortance des ports A, B, E, K (MEBI)

$000D	RDPK	0	0	RDPE	0	0	RDPB	RDPA	**RDRIV**
Reset	0	0	0	0	0	0	0	0	

1 / Toutes les lignes du port ont une sortance réduite. 0 / Pas de réduction sur les lignes du port. Sur la carte, on lit $00

Registre de contrôle de l'interface bus externe (MEBI)

$000E	0	0	0	0	0	0	0	ESTR	**EBICTL**
Reset	0	0	0	0	0	0	0	0	

Sans effet en mode single chip

Registre $000F réservé

Registre d'initialisation de la position de la RAM interne (MMC)

$0010	RAM15	RAM14	RAM13	RAM12	RAM11	0	0	RAMHAL	**INITRM**
Reset	0	0	0	0	1	0	0	1	

Les 5 bits poids forts initialisent l'adresse de base de la RAM interne. RAMHAL à 1 : la mémoire s'aligne sur le haut. à 0, sur le bas.
On lit $21 sur le kit

Registre d'initialisation de la position des registres internes (MMC)

$0011	0	REG14	REG13	REG12	REG11	0	0	0	**INITRG**
Reset	0	0	0	0	0	0	0	0	

Les 5 bits poids forts initialisent l'adresse de base du bloc des registres. On lit 0 sur le kit

Registre d'initialisation de la position des registres de l'EEPROM interne (MMC)

$0012	EE15	EE14	EE13	EE12	EE11	0	0	EEON	**INITEE**
Reset	0	0	0	0	0	0	0	1	

Les 5 bits poids forts initialisent l'adresse de base de l'EEPROM. EEON à 1 valide l'EEPROM. On lit $09 sur le kit

Registre de contrôle système « divers » (MMC)

$0013	0	0	0	0	EXSTR1	EXSTR0	ROMHM	ROMON	**MISC**
Reset	0	0	0	0	1	1	0	1	

EXSTR1 et EXSTR0 n'ont aucun sens en single chip (liés à l'étirement d'horloge)
ROMHM à 1 : ROM ou EEPROM seulement dans les adresses hautes (>$7FFF). ROMON à 1 : valide la ROM ou l'EEPROM

Registre de test réservé (MMC)

$0014	0	0	0	0	0	0	0	0	**MTST1**

Registres de contrôle des tests d'interrupton (INT) Pas utilisables en mode single chip

$0015	0	0	0	WRTINT	ADR3	ADR 2	ADR 1	ADR 0	**ITCR**
$0016	INTE	INTC	INTA	INT8	INT6	INT4	INT2	INT0	**ITEST**

Registre de test réservé (MMC)

$0017	0	0	0	0	0	0	0	0	**MTST0**

Registres $0018 et $0019 réservés

Identificateur de version

$001A	BIT15	BIT14	BIT13	BIT12	BIT11	BIT10	BIT9	BIT8	**PARTIDH**
$001B	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PARTIDL**

Registre 0 de taille mémoire (MMC)

$001C	reg_sw0	0	eeo_sw1	eep_sw0	0	ram_sw2	ram_sw1	ram_sw0	**MEMSIZ0**
Reset	-	-	-	-	-	-	-	-	

Registre d'état qui donne l'état des switchs de configuration de la mémoire système. On lit $25 sur le kit

eep_sw1:0	Espace alloué EEPROM
00	0K
01	2K
10	4K
11	8K

ram_sw 2:0	Espace alloué RAM	RAM mappable region	bits utilisés de INITRM
000	2K	2K	RAM15-RAM11
001	4K	4K	RAM15-RAM12
010	6K	8K	RAM15-RAM13
011	8K	8K	RAM15-RAM13
100	10K	16K	RAM15-RAM14

Registre 1 de taille mémoire (MMC)

$001D	rom_sw1	rom_sw0	0	0	0	0	pag_sw1	pag_sw0	**MEMSIZ1**
Reset	-	-	-	-	-	-	-	-	

Registre d'état qui donne l'état des switchs de configuration de la mémoire système. On lit $81 sur le kit

rom_sw1:0	Espace alloué Flash ou ROM
00	0K
01	16K
10	48K
11	64K

pag_sw1:0	Espace extérieur	Espace intérieur
00	876K	128K
01	768K	256K
10	512K	512K
11	0K	1M

Registre de contrôle IRQ pour définir le fonctionnement de la patte IRQ (MEBI)

$001E	IRQ	IRQEN	0	0	0	0	0	ESTR	**EBICTL**
Reset	0	0	0	0	0	0	0		

IRQE : à 1 / front descendant, à 0 / niveau bas
IRQEN : à 1 / patte externe IRQ connectée à la logique d'interruption

Interruption I de plus hautr priorité (INT)

$001F	PSEL7	PSEL6	PSEL5	PSEL4	PSEL3	PSEL2	PSEL1	0	**HPRIO**
Reset	1	1	1	1	0	0	1	0	

Les bits PSEL définissent l'interruption I la plus prioritaire (Par défaut IRQ)

Registres de $0020 à $0027 réservés

Registres de gestion des points d'arrêt (BKP)

$0028	BKEN	BKFULL	BKBDM	BKTAG	0	0	0	0	**BKPCT0**
$0029	BK0MBH	BK0MBL	BK1MBH	BK1MBL	BK0RWE	BK0RW	BK1RWE	BK1RW	**BKPCT1**
$002A	0	0	BK0V5	BK0V4	BK0V3	BK0V2	BK0V1	BK0V0	**BKP0X**
$002B	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	**BKP0H**
$002C	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	**BKP0L**
$002D	0	0	BK0V5	BK0V4	BK0V3	BK0V2	BK0V1	BK0V0	**BKP1X**
$002E	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	**BKP1H**
$002F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	**BKP1L**

Registre de page programme (MMC)

$0030	0	0	PIX5	PIX4	PIX3	PIX2	PIX1	PIX0	**PPAGE**
Reset	-	-	-	-	-	-	-	-	

On lit $38 sur le kit

Registre $0033 réservé

Registre de données 8 bits du Port K (MEBI)

$0032	PK7	PK6	PK5	PK4	PK3	PK2	PK1	PK0	**PORTK**
Reset	-	-	-	-	-	-	-	-	
Autres	ECS	XCS	XAB19	XAB18	XAB17	XAB16	XAB15	XAB14	

dans d'autres modes, bus d'adresses étendu (A19-A14). Pour ECS et XCS, voir MODE. Sur la carte, on lit $BF

Registre de direction de données 8 bits du Port K (MEBI)

$0033	DDRK7	DDRK6	DDRK5	DDRK4	DDRK3	DDRK2	DDRK1	DDRK0	**DDRK**
Reset	0	0	0	0	0	0	0	0	

DDRKx à 1 / configure PKx en sortie, à 0 / configure PKx en entrée

Registres attachés au CRG (Clock and Reset Generators)

$0034	0	0	SYN5	SYN4	SYN3	SYN2	SYN1	SYN0	**SYNR**
$0035	0	0	0	0	REFDV3	REFDV2	REFDV1	REFDV0	**REFDV**
$0036	0	0	0	0	0	0	0	0	**CTFLG**
$0037	RTIF	PORF	0	LOCKIF	LOCK	TRACK	SCMIF	SCM	**CRGFLG**
$0038	RTIE	0	0	LOCKIE	0	0	SCMIE	0	**CRGINT**
$0039	PLLSEL	PSTP	SYSWAI	ROAWAI	PLLWAI	CWAI	RTIWAI	COPWAI	**CLKSEL**
$003A	CME	PLLON	AUTO	ACQ	0	PRE	PCE	SCME	**PLLCTL**
$003B	0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0	**RTICTL**
$003C	WCOP	RSBCK	0	0	0	CR2	CR1	CR0	**COPCTL**
$003D	0	0	0	0	0	0	0	0	**FORBYP**
$003E	1	0	0	0	0	0	0	0	**CTCTL**
$003F	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**ARMCOP**

REGISTRES DU TIMER

Registre de sélection capture d'entrée/sortie sur comparaison du timer

$0040	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	**TIOS**
Reset	0	0	0	0	0	0	0	0	

IOSx à 1 / le canal x fonctionne en sortie sur comparaison. A 0 / le canal x fonctionne en capture d'entrée.

Registre de forçage de sortie sur comparaison

$0041	FOC7	FOC6	FOC5	FOC4	FOC3	FOC2	FOC1	FOC0	**CFORC**
Reset	0	0	0	0	0	0	0	0	

Force l'action de sortie sur comparaison sur les canaux 0 à 7.
L'écriture d'un 1 sur le bit x fait exécuter immédiatement l'action programmée pour la sortie sur comparaison N° x.

Registre de masquage de sortie sur comparaison 7 (?)

$0042	OC7M7	OC7M6	OC7M5	OC7M4	OC7M3	OC7M2	OC7M1	OC7M0	**OC7M**
Reset	0	0	0	0	0	0	0	0	

Voir notice détaillée.

Registre de données de la sortie sur comparaison 7

$0043	OC7D7	OC7D6	OC7D5	OC7D4	OC7D3	OC7D2	OC7D1	OC7D0	**OC7D**
Reset	0	0	0	0	0	0	0	0	

Voir notice détaillée.

Registre de comptage du timer

$0044	TCNT15	TCNT14	TCNT13	TCNT12	TCNT11	TCNT10	TCNT9	TCNT8	**TCNT**
$0045	TCNT7	TCNT6	TCNT5	TCNT4	TCNT3	TCNT2	TCNT1	TCNT0	
Reset	0	0	0	0	0	0	0	0	

Registre de contrôle N°1 timer système

$0046	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0	**TSCR1**
Reset	0	0	0	0	0	0	0	0	

TEN (Timer Enable) à 0 / le timer est dévalidé (économie d'énergie). à 1 / le timer fonctionne normalement
Si pour une raison quelconque, le timer n'est pas actif, il n'y a pas d'horloge ÷ 64 pour l'accumulateur d'impulsions
TSWAI à 0 / le timer continue pendant un Wait; à 1 / il s'arrête. TSFRZ à 0 / le timer continue en mode Freeze. à 1 / il s'arrête.
TFFCA (Timer Fast Flag Clear All). à 0 / l'abaissement des flags du timer se fait normalement par l'écriture d'un 1.
à1 / Pour TFLG1($4E), la lecture d'une capture d'entrée ou l'écriture d'une sortie sur comparaison abaisse le flag CnF correspondant
Pour TFLG2 ($4F), tout accès au registre TCNT ($44, $45) abaisse le flag TOF.
Tout accès aux registres PACN3 et PACN2 ($62, $63) abaisse les flags PAOVF et PAIF du registre PAFLG ($61).
Tout accès aux registres PACN1 et PACN0 ($64, $65) abaisse le flag PBOVF du registre PBFLG ($71).

Registre N°1 de bascule sur débordement du timer

$0047	TOV7	TOV6	TOV5	TOV4	TOV3	TOV2	TOV1	TOV0	**TTOV**
Reset	0	0	0	0	0	0	0	0	

à 1 / TOVx bascule la patte de sortie sur comparaison en cas de débordement. à 0 / pas de bascule.

Registre de contrôle N°1 du timer

$0048	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4	**TCTL1**
Reset	0	0	0	0	0	0	0	0	

Registre de contrôle N°2 du timer

$0049	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0	**TCTL2**
Reset	0	0	0	0	0	0	0	0	

OMx : Mode de sortie. OLx Niveau de sortie. Ces huit paires de bits de contrôle servent à définir l'action de sortie à exécuter à la suite d'une comparaison OCx réussie.
Quand l'un au moins des bits OMx ou OLx est à 1, la patte associée à OCx devient une sortie attachée à OCx.

OMx	OLx	Action
0	0	Timer déconnecté de la patte de sortie
0	1	Bascule de la ligne de sortie OCx
1	0	Mise à 0 de la ligne de sortie OCx
1	1	Mise à 1 de la ligne de sortie OCx

Registre de contrôle N°3 du timer

$004A	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A	**TCTL3**
Reset	0	0	0	0	0	0	0	0	

Registre de contrôle N°4 du timer

$004B	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A	**TCTL4**
Reset	0	0	0	0	0	0	0	0	

EDGxB, EDGxA Contrôle des frontsde capture d'entrée. Ces huit paires de bits de contrôle configurent les circuits détecteurs de front des captures d'entrée. Les quatre paires de bits de contrôle de TCTL4 configurent aussi les accumulateurs d'impulsions 8 bits PAC0 - 3.

EDGxB	EDGxA	Action
0	0	Capture dévalidée
0	1	Capture sur fronts montants
1	0	Capture sur fronts descendants
1	1	Capture sur fronts

Registre de validation des interruptions

\$004C	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	**TIE**
Reset	0	0	0	0	0	0	0	0	

A 1 / CxI valide une demande d'interruption sur montée du bit d'état CxF. à 0 / pas d'interruption

Registre de contrôle N°2 timer système

\$004D	TOI	0	0	0	TCRE	PR2	PR1	PR0	**TSCR2**
Reset	0	0	0	0	0	0	0	0	

TOI à 1 :demande d'interruption sur TOF TCRE (Timer Counter Reset Enable) à 1 :Reset du TCNT sur un OC7 réussi.

PR2, PR1, PR0 (Prescaler Select). Ces trois bits précisent le nombre de ÷2 à intercaler entre l'horloge du bus et le compteur principal

PR2	PR1	PR0	Facteur diviseur
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8

PR2	PR1	PR0	Facteur diviseur
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Registre d'état N°1 du timer principal

\$004E	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F	**TFLG1**
Reset	0	0	0	0	0	0	0	0	

CxF à 1 / occurence d'un événement (capture d'entrée ou sortie sur comparaison) sur la patte N°x.. On le remet à 0 en y écrivant un 1

Registre d'état N°2 du timer principal

\$004F	TOF	0	0	0	0	0	0	0	**TFLG2**
Reset	0	0	0	0	0	0	0	0	

TOF à 1 / Passage à 0 du compteur TCNT. On le remet à 0 en y écrivant un 1.

Registre N°0 de capture d'entrée/sortie sur comparaison

\$0050	TC015	TC014	TC013	TC012	TC011	TC010	TC09	TC08	**TC0**
\$0051	TC07	TC06	TC05	TC04	TC03	TC02	TC01	TC00	

Registre N°1 de capture d'entrée/sortie sur comparaison

\$0052	TC115	TC114	TC113	TC112	TC111	TC110	TC19	TC18	**TC1**
\$0053	TC17	TC16	TC15	TC14	TC13	TC12	TC11	TC10	

Registre N°2 de capture d'entrée/sortie sur comparaison

\$0054	TC215	TC214	TC213	TC212	TC211	TC210	TC29	TC28	**TC2**
\$0055	TC27	TC26	TC25	TC24	TC23	TC22	TC21	TC20	

Registre N°3 de capture d'entrée/sortie sur comparaison

\$0056	TC315	TC314	TC313	TC312	TC311	TC310	TC39	TC38	**TC3**
\$0057	TC37	TC36	TC35	TC34	TC33	TC32	TC31	TC30	

Registre N°4 de capture d'entrée/sortie sur comparaison

\$0058	TC415	TC414	TC413	TC412	TC411	TC410	TC49	TC48	**TC4**
\$0059	TC47	TC46	TC45	TC44	TC43	TC42	TC41	TC40	

Registre N°5 de capture d'entrée/sortie sur comparaison

\$005A	TC515	TC514	TC513	TC512	TC511	TC510	TC59	TC58	**TC5**
\$005B	TC57	TC56	TC55	TC54	TC53	TC52	TC51	TC50	

Registre N°6 de capture d'entrée/sortie sur comparaison

\$005C	TC615	TC614	TC613	TC612	TC611	TC610	TC69	TC68	**TC6**
\$005D	TC67	TC66	TC65	TC64	TC63	TC62	TC61	TC60	

Registre N°7 de capture d'entrée/sortie sur comparaison

\$005E	TC715	TC714	TC713	TC712	TC711	TC710	TC79	TC78	**TC7**
\$005F	TC77	TC76	TC75	TC74	TC73	TC72	TC71	TC70	

Registre de contrôle de l'accumulateur d'impulsions 16 bits A

\$0060	0	PAEN	PAMOD	PEDGE	CLK1	CLK0	PAOVI	PAI	**PACTL**
Reset	0	0	0	0	0	0	0	0	

L'accu 16-Bits PACA est formé en cascadant les deux accus 8-bits PACN3 et PACN2. Il partage la patte d'entrée (*PT7*) avec IC7.

PAEN :0 / L'accu 16 bits est dévalidé. PACN3 et PACN2 ont leur propre bit de validation dans ICPAR. PAIF est dévalidé.

1 / L'accu 16 bits est validé. Les bits de validation de PACN3 et PACN2 dans ICPAR sont sans effet. PAIF est validé.

PAMOD et PEDGE : Actif seulement si PAEN=1. Voir tableau ci-contre.

CLK1, CLK0 **:** Voir tableau ci-contre. Si l'accu est dévalidé (PAEN = 0), l'horloge du pré-diviseur du timer sert d'horloge d'entrée du compteur du timer.

PAOVI : Demande d'interruption sur PAOVF : 0 / non transmise. 1 / transmise.

PAI **:** Demande d'interruption sur PAIF : 0 / non transmise. 1 / transmise.

PAMOD	PEDGE	Pin Action
0	0	Front descendant
0	1	Front montant
1	0	Horloge/64 validée au niveau haut
1	1	Horloge/64 validée au niveau bas

CLK1	CLK0	Horloge Source
0	0	Horloge du prédiviseur timer
0	1	PACLK
1	0	PACLK/256
1	1	PACLK/65536

Registre d'indicateurs de l'accumulateur d'impulsions 16 bits A

$0061	0	0	0	0	0	0	PAOVF	PAIF	**PAFLG**
Reset	0	0	0	0	0	0	0	0	

PAOVF : mis si PACA (16 bits) passe de $FFFF à 0, ou si PAC3 (8 bits) passe de $FF à 0. On le remet à 0 en y écrivant un 1
PAIF : Mis quand le front choisi est détecté sur la patte *PT7*. En mode événement, le front événement déclenche PAIF, et en chronomètre, le front de fin de porte déclenche PAIF. On le remet à 0 en y écrivant un 1

Registre de comptage N°3 de l'accumulateur d'impulsions

$0062	PACNT7/15	PACNT6/14	PACNT5/13	PACNT4/12	PACNT3/11	PACNT2/10	PACNT1/9	PACNT0/8	**PACN3**
Reset	0	0	0	0	0	0	0	0	

Registre de comptage N°2 de l'accumulateur d'impulsions

$0063	PACNT7	PACNT6	PACNT5	PACNT4	PACNT3	PACNT2	PACNT1	PACNT0	**PACN2**
Reset	0	0	0	0	0	0	0	0	

Les deux accus 8 bits PACN3 et PACN2 sont cascadés pour former l'accu 16 bits PACA. Quand PACA est validé (bit PAEN duregistre PACTL), PACN3 et PACN2 sont respectivement l'octet fort et l'octet faible de PACA.
Quand PACN3 déborde de $FF à 0, l'indicateur PAOVF du registre PAFLG passe à 1.

Registre de comptage N°1 de l'accumulateur d'impulsions

$0064	PACNT7/15	PACNT6/14	PACNT5/13	PACNT4/12	PACNT3/11	PACNT2/10	PACNT1/9	PACNT0/8	**PACN1**
Reset	0	0	0	0	0	0	0	0	

Registre de comptage N°0 de l'accumulateur d'impulsions

$0065	PACNT7	PACNT6	PACNT5	PACNT4	PACNT3	PACNT2	PACNT1	PACNT0	**PACN0**
Reset	0	0	0	0	0	0	0	0	

Les deux accus 8 bits PACN1 et PACN0 sont cascadés pour former l'accu 16 bits PACB. Quand PACB est validé (bit PBEN duregistre PBCTL), PACN1et PACN0 sont respectivement l'octet fort et l'octet faible de PACB.
Quand PACN1 déborde de $FF à 0, l'indicateur PBOVF du registre PBFLG passe à 1.

Registre de contrôle du module décomptage 16 bits.

$0066	MCZI	MODMC	RDMCL	ICLAT	FLMC	MCEN	MCPR1	MCPR0	**MCCTL**
Reset	0	0	0	0	0	0	0	0	

MCZI : Demande d'interruption sur MCZF. 1 / Demande transmise. 0 / Demande non transmise
MODMC :0 / Le compteur décompte depuis la valeur chargée. Si 0, il s'arrête à 0. Si 1 / Il boucle de 0 à la valeur choisie
RDMCL Une lecture du compteur du module ramène: 0 / la valeur courante. 1 / La valeur chargée.
ICLAT : En mode latch de capture, l'écriture d'un 1 dans ce bit sauvegarde les registres de capture d'entrée TC0 à TC3 et les accus 8 bits correspondants et met à 0 les accus. L'écriture d'un 0 sur ce bit n'a pas d'effet, et sa lecture donne toujours 0.
FLMC L'écriture d'un 1 dans ce bit réinitialise le registre de décomptage à sa valeur de chargement. Elle réinitialise également le pré-diviseur. L'écriture d'un 0 sur ce bit n'a pas d'effet, et sa lecture donne 0.
MCEN : 0 / Compteur de module invalidé. 1 / Compteur de module validé.
MCPR1, MCPR0 **:** Sélection du pré-diviseur du module. Voir tableau ci-contre.
La nouvelle valeur prend effet après rechargement du compteur du module.

MCPR1	MCPR0	Valeur du pré-diviseur
0	0	1
0	1	4
1	0	8
1	1	16

Registre d'état du module décomptage 16 bits.

$0067	MCZF	0	0	0	POLF3	POLF2	POLF1	POLF0	**MCFLG**
Reset	0	0	0	0	0	0	0	0	

MCZF : Mis quand le décompteur arrive à 0. On le remet à 0 en y écrivant un 1. L'écriture d'un 0 est sans effet.
Le bit POLFx correspond à l'entrée timer *PTx*. Il in dique que la première capture est due à un front 0 / descendant. 1 / montant.

Registre de contrôle d'entrée des accumulateurs d'impulsions.

$0068	0	0	0	0	PA3EN	PA2EN	PA1EN	PA0EN	**ICPAR**
Reset	0	0	0	0	0	0	0	0	

On ne peut valider les accus 8-bits PAC3 et PAC2 que si le bit PAEN de PATCL est à 0. Sinon, PA3EN et PA2EN sont sans effet.
On ne peut valider les accus 8-bits PAC1 et PAC0 que si le bit PBEN de PBTCL est à 0. Sinon, PA1EN et PA0EN sont sans effet.
PAxEN : 0 / Accu 8 bits N°x dévalidé. 1 / Accu 8 bits N°x validé.

Registre de contrôle du compteur de délai.

$0069	0	0	0	0	0	0	DLY1	DLY0	**DLYCT**
Reset	0	0	0	0	0	0	0	0	

Après détection d'un front actif sur une patte de capture d'entrée, le compteur de délai attend un certain nombre de cycles machine avant d'envoyer une impulsion en sortie.

DLY1	DLY0	Délai
0	0	Dévalidé
0	1	256 cycles d'horloge
1	0	512 cycles d'horloge
1	1	1024 cycles d'horloge

Registre de contrôle d'écrasement des captures d'entrée

$006A	NOVW7	NOVW6	NOVW5	NOVW4	NOVW3	NOVW2	NOVW1	NOVW0	**ICOVW**
Reset	0	0	0	0	0	0	0	0	

NOVWx : 0 / Pas de précaution contre l'écrasement. 1 / On ne peut écrire dans un registre que s'il est vide (déjà lu)

Registre de contrôle du système en entrée (?)

\$006B	SH37	SH26	SH15	SH04	TFMOD	PACMX	BUFEN	LATQ	**ICSYS**
Reset	0	0	0	0	0	0	0	0	

Voir notice détaillée.

Registre de test du timer

\$006D	0	0	0	0	0	0	TCBYP	0	**TIMTST**
Reset	0	0	0	0	0	0	0	0	

Pour tests uniquement.

Registre de contrôle de l'accumulateur d'impulsions 16 bits B

\$0070	0	PBEN	0	0	0	0	PBOVI	0	**PBCTL**
Reset	0	0	0	0	0	0	0	0	

L'accu 16-Bits PACB est formé en cascadant les deux accus 8-bits PACN1 et PACN0. Il partage la patte d'entrée (*PT7*) avec IC0.
PBEN :0 / L'accu 16 bits est dévalidé. PACN1 et PACN0 ont leur propre bit de validation dans ICPAR.
1 / L'accu 16 bits est validé. Les bits de validation de PACN1 et PACN0 dans ICPAR sont sans effet.
PBOVI Validation d'interruption sur débordement (PBOVF). 1 / demande transmise. 0 / demande non transmise

Registre d'indicateurs de l'accumulateur d'impulsions 16 bits B

\$0071	0	0	0	0	0	0	PBOVF	0	**PBFLG**
Reset	0	0	0	0	0	0	0	0	

PBOVF : mis si PACB (16 bits) passe de \$FFFF à 0, ou si PAC1 (8 bits) passe de \$FF à 0. On le remet à 0 en y écrivant un 1

Registre N°3 de rangement d'accumulateur 8 bits

\$0072	PA3H7	PA3H6	PA3H5	PA3H4	PA3H3	PA3H2	PA3H1	PA3H0	**PA3H**
Reset	0	0	0	0	0	0	0	0	

Registre N°2 de rangement d'accumulateur 8 bits

\$0073	PA2H7	PA2H6	PA2H5	PA2H4	PA2H3	PA2H2	PA2H1	PA2H0	**PA2H**
Reset	0	0	0	0	0	0	0	0	

Registre N°1 de rangement d'accumulateur 8 bits

\$0074	PA1H7	PA1H6	PA1H5	PA1H4	PA1H3	PA1H2	PA1H1	PA1H0	**PA1H**
Reset	0	0	0	0	0	0	0	0	

Registre N°0 de rangement d'accumulateur 8 bits

\$0075	PA0H7	PA0H6	PA0H5	PA0H4	PA0H3	PA0H2	PA0H1	PA0H0	**PA0H**
Reset	0	0	0	0	0	0	0	0	

Registre de comptage du module décomptage 16 bits

\$0076	MCCNT15	MCCNT14	MCCNT13	MCCNT12	MCCNT11	MCCNT10	MCCNT9	MCCNT8	**MCCNT**
\$0077	MCCNT7	MCCNT6	MCCNT5	MCCNT4	MCCNT3	MCCNT2	MCCNT1	MCCNT0	
Reset	1	1	1	1	1	1	1	1	

Voir notice détaillée

Registre de rangement N°0 de capture d'entrée du timer

\$0078	TC15	TC14	TC13	TC12	TC11	TC10	TC9	TC8	**TCOH**
\$0079	TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0	
Reset	0	0	0	0	0	0	0	0	

Registre de rangement N°1 de capture d'entrée du timer

\$007A	TC15	TC14	TC13	TC12	TC11	TC10	TC9	TC8	**TC1H**
\$007B	TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0	
Reset	0	0	0	0	0	0	0	0	

Registre de rangement N°2 de capture d'entrée du timer

\$007C	TC15	TC14	TC13	TC12	TC11	TC10	TC9	TC8	**TC2H**
\$007D	TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0	
Reset	0	0	0	0	0	0	0	0	

Registre de rangement N°3 de capture d'entrée du timer

\$007E	TC15	TC14	TC13	TC12	TC11	TC10	TC9	TC8	**TC3H**
\$007F	TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0	
Reset	0	0	0	0	0	0	0	0	

CONVERTISSEUR ANALOGIQUE-NUMERIQUE ATD0

Registre de contrôle N°0 de l'ATD0

$0080	0	0	0	0	0	0	0	0	**ATD0CTL0**
Reset	0	0	0	0	0	0	0	0	

Réservé

Registre de contrôle N°1 de l'ATD0

$0081	0	0	0	0	0	0	0	0	**ATD0CTL1**
Reset	0	0	0	0	0	0	0	0	

Réservé

Registre de contrôle N°2 de l'ATD0

$0082	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIGE	ASCIE	ASCIF	**ATD0CTL2**
Reset	0	0	0	0	0	0	0	0	

ADPU (ATD Power Up): alimentation du convertisseur
AFFC : CCF s'abaisse à la lecture du résultat [1/sans, 0/avec] lecture préalable du status
AWAI : en mode wait :1/l'alimentation est coupée, 0/le convertisseur reste alimenté
ETRIGLE et ETRIGP fixent le signal déclenchant sur le canal 7.
ETRIGE : valide le déclenchement sur le canal 7. la mesure sur ce canal n'a pas de sens
ASCIE : validation d'inter sur séquence complète. ASCIF : indicateur associé

ETRIGLE	ETRIGP	Signal déclenchant
0	0	front descendant
0	1	front montant
1	0	bas niveau
1	1	haut niveau

Registre de contrôle N°3 de l'ATD0

$0083	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0	**ATD0CTL3**
Reset	0	0	1	0	0	0	0	0	

S8C, S4C, S2C et S1C définissent le nombre Nc de conversions par séquence (cf tableau ci-contre)
FIFO : 0/Les conversions sont rangées à partir du 1er registre
: 1/ Les conversions sont rangées à la suite
FRZ1, FRZ0. Possibilité de gel de l'ATD sur point d'arrêt

S8C	S4C	S2C	S1C	NC
0	0	0	0	8
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4

S8C	S4C	S2C	S1C	NC
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	X	X	X	8

Registre de contrôle N°4 de l'ATD0

$0084	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0	**ATD0CTL4**
Reset	0	0	0	0	0	1	0	1	

SRES8 : 1/8 bits, 0/10 bits
SMP1 et SMP0 définissent la longueur 2^{e} phase d'échantillonnage
Les 5 bits suivant définissent le pré-diviseur PRS. La fréquence de l'horloge de conversion est calculée à partir de l'horloge du bus E par la formule E×0.5/(PRS+1)

SMP1	SMP0	Longueur 2^{e} phase d'échantillonnage
0	0	2 périodes d'horloge de conversion A/D
0	1	4 périodes d'horloge de conversion A/D
1	0	8 périodes d'horloge de conversion A/D
1	1	16 périodes d'horloge de conversion A/D

Registre de contrôle N°5 de l'ATD0

$0085	DJM	DSGN	SCAN	MULT	0	CC	CB	CA	**ATD0CTL5**
Reset	0	0	0	0	0	0	0	0	

DJM : Résultats justifiés :1/à droite, 0/ à gauche DSGN : Résultats en représentation 1/signée, 0/non signée
SCAN : 1/séquences continues de conversion, 0/une seule séquence
MULT : La séquence de conversion concerne : 0/un seul canal, 1/plusieurs canaux

Registre d'état N°0 de l'ATD0

$0086	SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0	**ATDSTAT0**
Reset	0	0	0	0	0	0	0	0	

SCF est mis à la fin d'une séquence de conversion. Si elles s'enchaînent (SCAN=1), il est mis à la fin de chaque séquence. Remis à 0 dans les cas suivants : Ecriture d'un"1" sur SCF, écriture dans ATDCTL5 (nouvelle séquence), lecture d'un résultat si AFFC=1.
ETORF est mis en mode déclenchement par front (ETRIGLE=0) si on détecte d'autres fronts pendant la conversion. Remis à 0 dans les cas suivants : Ecriture d'un '1' sur ETORF, écriture sur ATDCTL2, ATDCTL3, ATDCTL4 ou ATDCTL5.
FIFOR indique qu'on a écrit dans un registre résultat avant effacement de son bit de conversion complète (CCF) Ce flag descend à la suite d'une des actions suivantes : Ecriture d'un '1' sur FIFOR, écriture sur ATDCTL5 (nouvelle séquence)
CC2, CC1, CC0 : Compteur de conversions qui pointe sur le registre destinataire de la conversion en cours. En mode FIFO (FIFO=1), le compteur n'est pas initialisé et reboucle sur lui même.Sinon le compteur est mis à 0 en début et en fin de séquence de conversion.

Registre réservé

$0088	0	0	0	0	0	0	0	0	**ATDTEST0**
Reset	0	0	0	0	0	0	0	0	

Registre1 de test de l'ATD0

$0089	0	0	0	0	0	0	0	SC	**ATDTEST1**
Reset	0	0	0	0	0	0	0	0	

Registre d'état N°1 de l'ATD0

$008B	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0	**ATDSTAT1**
Reset	0	0	0	0	0	0	0	0	

CCFx : Indique la fin de la conversion N°x (x=7,6,5,4,3,2,1,0) 1 / Fin de la conversion N° x. 0 / Conversion N° x pas terminée
Il passe à 1 quand la x^{e} conversion a été rangée dans le registre ATDDRx. Remis à 0 dans les cas suivants : Ecriture sur ATDCTL5 (nouvelle séquence), Si AFFC=0, lecture de ATDSTAT1 suivie de celle de ATDDRx, si AFFC=1, lecture de ATDDRx.

Registre de validation d'entrée de l'ATD0

$008D	IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1	IEN0	**ATD0IEN**
Reset	0	0	0	0	0	0	0	0	

IENx : Validation de la patte *PTADx* comme entrée logique du port AD. 1 / Entrée logique validée. 0 / Entrée analogique validée.

Registre de port de donnée de l'ATD0

$008F	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0	**PORTAD**
Reset	1	1	1	1	1	1	1	1	
Fonction	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	

Si la patte *PADx* est validée comme entrée logique (IENx=1), PTADx indique le niveau logique de la patte. Sinon il est à 1.

Registres 0 de résultats de conversion de l'ATD0, justifié à gauche

$0090	BIT9	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	**ATD0DR0H**
$0091	BIT1	BIT0	0	0	0	0	0	0	**ATD0DR0L**
Reset	-	-	-	-	-	-	-	-	
	-	-	0	0	0	0	0	0	

Registres 0 de résultats de conversion de l'ATD0, justifié à droite

$0090	0	0	0	0	0	0	BIT9	BIT8	**ATD0DR0H**
$0091	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**ATD0DR0L**
Reset	0	0	0	0	0	0	-	-	
	-	-	-	-	-	-	-	-	

Dans les adresses suivantes, de $0092 à $009F, on retrouve les 7 autres registres de résultats avec les deux justifications

Registres de résultats de conversion de l'ATD0

$0090	BIT9	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	**ATD0DR0H**
$0091	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**ATD0DR0L**
$0092	BIT9	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	**ATD0DR1H**
$0093	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**ATD0DR1L**
$0094	BIT9	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	**ATD0DR2H**
$0095	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**ATD0DR2L**
$0096	BIT9	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	**ATD0DR3H**
$0097	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**ATD0DR3L**
$0098	BIT9	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	**ATD0DR4H**
$009A	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**ATD0DR5L**
$009B	BIT9	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	**ATD0DR5H**
$009C	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**ATD0DR6L**
$009D	BIT9	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	**ATD0DR6H**
$009E	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**ATD0DR7L**
$009F	BIT9	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	**ATD0DR7H**

REGISTRES DE GESTION DES PWM

$00A0	PWME7	PWME6	PWME5	PWME4	PWME3	PWME2	PWME1	PWME0	**PWME**
$00A1	PPOL7	PPOL6	PPOL5	PPOL4	PPOL3	PPOL2	PPOL1	PPOL0	**PWMPOL**
$00A2	PCLK7	PCLK6	PCLK5	PCLK4	PCLK3	PCLK2	PCLK1	PCLK0	**PWMCLK**
$00A3	0	PCKB2	PCKB1	PCKB0	0	PCKA2	PCKA1	PCKA0	**PWMPRCLK**
$00A4	CAE7	CAE6	CAE5	CAE4	CAE3	CAE2	CAE1	CAE0	**PWMCAE**
$00A5	CON67	CON45	CON23	CON01	PSWAI	PFRZ	0	0	**PWMCTL**
$00A6	0	0	0	0	0	0	0	0	**PWMTST1**
$00A7	0	0	0	0	0	0	0	0	**PWMPRSC2**
$00A8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMSCLA**
$00A9	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMSCLB**
$00AA	0	0	0	0	0	0	0	0	**PWMSCNTA3**
$00AB	0	0	0	0	0	0	0	0	**PWMSCNTB4**
$00AC	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMCNT0**
$00AD	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMCNT1**
$00AE	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMCNT2**
$00AF	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMCNT3**
$00B0	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMCNT4**
$00B1	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMCNT5**
$00B2	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMCNT6**
$00B3	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMCNT7**
$00B4	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMPER0**
$00B5	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMPER1**
$00B6	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMPER2**
$00B7	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMPER3**
$00B8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMPER4**
$00B9	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMPER5**
$00BA	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMPER6**
$00BB	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMPER7**
$00BC	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMDTY0**
$00BD	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMDTY1**
$00BE	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMDTY2**
$00BF	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMDTY3**
$00C0	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMDTY4**
$00C1	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMDTY5**
$00C2	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMDTY6**
$00C3	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**PWMDTY7**
$00C4	PWMIF	PWMIE	RSTRT	LVL	0	7IN	7INL	7EN	**PWMSDN**

Registres de $00C5 à $00C7 réservés

INTERFACE SERIE ASYNCHRONE (SCI0, SCI1)

Registre de bauds SCI0

$00C8	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8	**SCI0BDH**
$00C9	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0	**SCI0BDL**

SBR12 - SBR0 : Le nombre BR défini par ces 13 bits détermine le nombre de bauds du SCI selon : baud = horloge/(16 × BR).
Le générateur de bauds est dévalidé jusqu'à ce que le bit TE ou RE soit mis pour la première fois. On le dévalide si BR=0.
Une écriture sur SCIBDH met la donnée en attente. Elle n'a d'effet que suivie d'une écriture sur SCIBDL.

Registre de contrôle N°1 de SCI0

$00CA	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT	**SCI0CR1**

LOOPS et RSRC définissent le mode bouclé selon le tableau ci-contre. Pour plus de détails, se reporter à la notice du SCI.

LOOPS	RSRC	Fonction
0	x	Fonctionnement normal
1	0	Mode bouclé avec entrée Rx reliée intérieurement à sortie Tx
1	1	Mode simple fil avec entrée Rx connectée à **TXD**

SCISWAI : Arrête ou non le SCI en mode wait. 1 / SCI arrêté. 0 / SCI validé.
M : Format de données. 1 / 1 start bit, 9 bits de données, 1 stop bit. 0 / 1 start bit, 8 bits de données, 1 stop bit
WAKE : Détermine la condition de réveil du SCI : 1 / Marque d'adresse (Address mark). 0 / Ligne oisive (Idle line)
ILT : Détermine quand le récepteur commence à compter les '1' logiques comme bits de caractères repos.
1 / Le décompte commence après le stop bit. 0 / Le décompte commence après le stop bit
PE : Valide la fonction parité (insère un bit de parité en MSB. 1 / Parité validée. 0 / Pas de parité
PT Choisit le type de parité 1 / Parité impaire. 0 / Parité paire.

Registre de contrôle N°2 de SCI0

$00CB	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	**SCI0CR2**

TIE : Valide les interruptions sur montée du bit d'état TDRE : 1 / Demande validée. 0 / Demande non transmise
TCIE : Valide les interruptions sur montée du bit d'état TC : 1 / Demande validée. 0 / Demande non transmise
RIE : Valide les interruptions sur montée du bit d'état RDRF : 1 / Demande validée. 0 / Demande non transmise
ILIE : Valide les interruptions sur montée du bit d'état IDLE : 1 / Demande validée. 0 / Demande non transmise
TE : Valide le transmetteur du SCI et configure la patte *Port S1* comme *TXD*. 1 / Transmetteur validé. 0 / Transmetteur invalidé.
RE : Valide le récepteur du SCI et configure la patte *Port S0* comme *RXD*. 1 / Récepteur validé. 0 / Récepteur invalidé.
RWU : Bit de réveil du receveur (Receiver Wakeup Bit)
SBK : Bit d'envoi d'un break (Send Break Bit)

Registre d'état N°1 de SCI0

$00CC	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	**SCI0SR1**

TDRE (Transmit Data Register Empty) est mis quand le registre à décalage reçoit un octet du registre de données du SCI. Quand le bit est à 1,le registre SCIDRH/L est vide et peut recevoir une nouvelle valeur à transmettre.
TC (Transmit Complete) est à l'état bas si une transmission est en cours, à l'état haut sinon.
RDRF (Receive Data Register Full) est mis quand l'octet reçu est transféré du registre à décalage vers le registre de données du SCI.
IDLE (Idle Line = ligne oisive) est mis quand 10 (ou 11) bits '1' consécutifs apparaissent sur le récepteur.
OR (Overrun) est mis si le logiciel ne peut lire SCIDR avant l'arrivée d'une nouvelle donnée dans le registre à décalage..
NF (Noise) est mis quand le SCI détecte du bruit sur le récepteur.
FE (Framing Error) est mis quand on rencontre un stop bit à 0.
PF (Parité) est mis quand la parité est validée (bit PE de SCICR1) si la donnée reçue n'a pas la bonne parité.
Note : on remet tous ces bits à 0 par une lecture de SCISR1 suivie d'une écriture(TDRE,TC)ou d'une lecture(les autres)sur SCIDRL

Registre d'état N°2 de SCI0

$00CD	0	0	0	0	0	BRK13	TXDIR	RAF	**SCI0SR2**

BRK13 (Longueur du break en transmission) 1 / Longueur du break 13 ou 14. 0 / longueur du break 10 ou 11
TXDIR Direction de la patte TXD en mode simple fil. 1 / sortie, 0 / entrée
RAF (Receiver Active) est mis si une réception est en cours.

Données hautes de SCI0

$00CE	R8	T8	0	0	0	0	0	0	**SCI0DRH**

R8 est le neuvième bit de donnée en réception quand le SCI est configuré avec 9 bits de données (M = 1).
T8 est le neuvième bit de donnée en transmission quand le SCI est configuré avec 9 bits de données (M = 1).

Données basses de SCI0

$00CF	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0	**SCI0DRL**

Registres de SCI1 analogues à SCI0

$00D0	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8	**SCI1BDH**
$00D1	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0	**SCI1BDL**
$00D2	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT	**SCI1CR1**
$00D3	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	**SCI1CR2**
$00D4	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	**SCI1SR1**
$00D5	0	0	0	0	0	BRK13	TXDIR	RAF	**SCI1SR2**
$00D6	R8	T8	0	0	0	0	0	0	**SCI1DRH**
$00D7	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0	**SCI1DRL**

INTERFACE SERIE SYNCHRONE (SPI0)

Registre de contrôle N°1 du SPI0

$00D8	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE	**SPICR1**

SPIE : Valide les interruptions sur montée des bits d'état SPIF ou MODF. 1 / Demande validée. 0 / Demande non transmise
SPE Valide le système SPI et lui attribue les pattes *PortS4:7*. 1 / Système SPI validé. 0 / Système dévalidé.
SPTIE : Valide les interruptions sur montée du bit d'état SPTEF. 1 / Demande validée. 0 / Demande non transmise
MSTR à 1 / Mode maître. à 0 / Mode esclave
CPOL (polarité d'horloge) 1 / active niveau bas. 0 / active niveau haut.
CPHA : Ce bit sert à déphaser l'horloge série SCK. 1 / le premier front de SCK a lieu au début des huit cycles de transfert. 0 / le premier front de SCK a lieu un demi-cycle après le début du transfert.
SSOE Validation de sortie Slave Select. Voir tableau ci-contre
LSBFE 1 / Donnée transmise LSB en premier.0 / Donnée transmise MSB en premier.

MOD FEN	SSOE	Mode Maitre	Mode Esclave
0	0	SS pas utilisé par SPI	entrée SS
0	1	SS pas utilisé par SPI	entrée SS
1	0	entrée SS avec MODF	entrée SS
1	1	sortie SS	entrée SS

Registre de contrôle N°2 du SPI0

$00D9	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPCO	**SPICR2**

MODFEN.1 / Autorise la montée de l'indicateur MODF. 0 / Dévalide l'indicateur.
BIDIROE 1 / Buffer de sortie validé. 0 / Buffer de sortie dévalidé
SPISWAI 1 / Stoppe l'horloge SPI en mode wait. 0 / l'horloge SPI fonctionne normalement en mode wait.
SPC0 . Avec le bit de contrôle MSTR, valide les configurations bidirectionnelles des pattes selon le tableau ci-contre.

	Pin Mode	SPC 0	MSTR	MISO [1]	MOSI [2]	SCK [3]	SS [4]
A	Normal	0	0	Slave Out	Slave In	SCK in	SS in
B	Normal	0	1	Master In	Master Out	SCK out	SS I/O
C	Bidirectional	1	0	Slave I/O	—	SCK in	SS In
D	Bidirectional	1	1	—	Master I/O	SCK out	SS I/O

NOTES:
1. Sortie esclave valide si BIDIROE=1, SS=0, MSTR=0 (C)
2. Sortie maître valide si BIDIROE=1 et MSTR=1 (D)
3. Sortie SCK valide si MSTR=1 (B, D)
4. Sortie SS valide si MODFEN=1, SSOE=1, MSTR=1(B, D)

Registre de bauds du SPI0

$00DA	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0	**SPIBR**

Nombre de bauds = Horloge / Diviseur de bauds. Diviseur de bauds = (SPPR+1) × 2exp(SPR+1)

Registre d'état du SPI0

$00DB	SPIF	0	SPTEF	MODF	0	0	0	0	**SPISR**

SPIF . 1 / Il y a une nouvelle donnée dans SPIDR. 0 / Transfert pas encore terminé
SPTEF 1 / Registre de données SPI vide. 0 / Registre de données SPI non vide
MODF Faute de mode. 1 / Une faute de mode s'est produite. 0 / Pas de faute de mode.

Registre $00DC réservé

Registre de données du SPI0

$00DD	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0	**SPIDR**

Ce registre est à la fois registre d'entrée et de sortie des données du SPI. Une écriture dans ce registre met un octet en attente de transmission. Si le SPI est configuré en maître, la donnée est transmise immédiatement après la fin de la transmission en cours. L'indicateur SPTEF du registre d'état indique quand le registre de données peut recevoir une nouvelle donnée.

Registres $00DE et $00DF réservés

ICI MANQUENT LES REGISTRES RELATIFS A
IIC,
BDLC,
SPI1 et SPI2,
MEMOIRES FLASH (FES) ET EEPROM (EETS)
LE SECOND CONVERTISSEUR (ATD1)
BUS CAN (CAN0, CAN1, CAN2 et CAN3)

PORTS D'ENTREE SORTIE LOGIQUES (PIM)

Registre d'entrée sortie du port T

$0240	PTT7	PTT6	PTT5	PTT4	PTT3	PTT2	PTT1	PTT0	**PTT**
Reset	0	0	0	0	0	0	0	0	
Timer	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	

Si le bit de direction de donnée de la patte est à 1, une lecture donne la valeur du registre de port, sinon elle donne la valeur de la patte.

Registre d'entrée du port T

$0241	PTIT7	PTIT6	PTIT5	PTIT4	PTIT3	PTIT2	PTIT1	PTIT0	**PTIT**
Reset	-	-	-	-	-	-	-	-	

L'écriture dans ce registre est sans effet. Ce registre donne toujours l'état des pattes associées.

Registre de direction de données du port T

$0242	DDRT7	DDRT6	DDRT5	DDRT4	DDRT3	DDRT2	DDRT1	DDRT0	**DDRT**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure la direction de chaque patte du port T. DDRTx à 1/ patte x en sortie. DDRTx à 0/ patte x en entrée

Registre de réduction de sortance du port T

$0243	RDRT7	RDRT6	RDRT5	RDRT4	RDRT3	RDRT2	RDRT1	RDRT0	**RDRT**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure la sortance de chaque patte en sortie . RDRTx à 1/ sortance réduite au tiers. RDRTx à 0 /sortance complète.

Registre de validation du pull du port T

$0244	PERT7	PERT6	PERT5	PERT4	PERT3	PERT2	PERT1	PERT0	**PERT**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure l'activation d'un pull-up ou d'un pull-down en entrée. PERTx à 1/ pull activé. PERTx à 0/pull désactivé.

Registre de choix de polarité du port T

$0245	PPST7	PPST6	PPST5	PPST4	PPST3	PPST2	PPST1	PPST0	**PPST**
Reset	0	0	0	0	0	0	0	0	

Ce registre choisit entre pull-up et pull-down. PPSTx à 1/ pull-down. PPSTx à 0/ pull-up.

Registre d'entrée sortie du port S

$0248	PTS7	PTS6	PTS5	PTS4	PTS3	PTS2	PTS1	PTS0	**PTS**
Reset	0	0	0	0	0	0	0	0	
SPI/SCI	SS0	SCK0	MOSI0	MISO0	TXD1	RXD1	TXD0	RXD0	

Si le bit de direction de donnée de la patte est à 1, une lecture donne la valeur du registre de port, sinon elle donne la valeur de la patte.

Registre d'entrée du port S

$0249	PTIS7	PTIS6	PTIS5	PTIS4	PTIS3	PTIS2	PTIS1	PTIS0	**PTIS**
Reset	-	-	-	-	-	-	-	-	

L'écriture dans ce registre est sans effet. Ce registre donne toujours l'état des pattes associées.

Registre de direction de données du port S

$024A	DDRS7	DDRS6	DDRS5	DDRS4	DDRS3	DDRS2	DDRS1	DDRS0	**DDRS**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure la direction de chaque patte du port S. DDRSx à 1/ patte x en sortie. DDRSx à 0/ patte x en entrée

Registre de réduction de sortance du port S

$024B	RDRS7	RDRS6	RDRS5	RDRS4	RDRS3	RDRS2	RDRS1	RDRS0	**RDRS**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure la sortance de chaque patte en sortie . RDRSx à 1/ sortance réduite au tiers. RDRSx à 0 /sortance complète.

Registre de validation du pull du port S

$024C	PERS7	PERS6	PERS5	PERS4	PERS3	PERS2	PERS1	PERS0	**PERS**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure l'activation d'un pull-up ou d'un pull-down en entrée. PERSx à 1/ pull activé. PERSx à 0/pull désactivé.

Registre de choix de polarité du port S

$024D	PPSS7	PPSS6	PPSS5	PPSS4	PPSS3	PPSS2	PPSS1	PPSS0	**PPSS**
Reset	0	0	0	0	0	0	0	0	

Ce registre choisit entre pull-up et pull-down. PPSSx à 1/ pull-down. PPSSx à 0/ pull-up.

Registre de mode ou-câblé du port S

$024E	WOMS7	WOMS6	WOMS5	WOMS4	WOMS3	WOMS2	WOMS1	WOMS0	**WOMS**
Reset	0	0	0	0	0	0	0	0	

WOMSx à 1 / sortie x collecteur ouvert. à 0 / sortie normale.

Registre d'entrée sortie du port M

$0250	PTM7	PTM6	PTM5	PTM4	PTM3	PTM2	PTM1	PTM0	**PTM**
Reset	0	0	0	0	0	0	0	0	
CAN	TXCAN3	RXCAN3	TXCAN2	RXCAN2	TXCAN1	RXCAN1	TXCAN0	RXCAN0	
BDLC							TXB	RXB	

Si le bit de direction de donnée de la patte est à 1, une lecture donne la valeur du registre de port, sinon elle donne la valeur de la patte.

Registre d'entrée du port M

$0251	PTIM7	PTIM6	PTIM5	PTIM4	PTIM3	PTIM2	PTIM1	PTIM0	**PTIM**
Reset	0	0	0	0	0	0	0	0	

L'écriture dans ce registre est sans effet. Ce registre donne toujours l'état des pattes associées.

Registre de direction de données du port M

$0252	DDRM7	DDRM6	DDRM5	DDRM4	DDRM3	DDRM2	DDRM1	DDRM0	**DDRM**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure la direction de chaque patte du port M. DDRMx à 1/ patte x en sortie. DDRMx à 0/ patte x en entrée

Registre de réduction de sortance du port M

$0253	RDRM7	RDRM6	RDRM5	RDRM4	RDRM3	RDRM2	RDRM1	RDRM0	**RDRM**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure la sortance de chaque patte en sortie . RDRMx à 1/ sortance réduite au tiers. RDRMx à 0 /sortance complète.

Registre de validation du pull du port M

$0254	PERM7	PERM6	PERM5	PERM4	PERM3	PERM2	PERM1	PERM0	**PERM**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure l'activation d'un pull-up ou d'un pull-down en entrée. PERMx à 1/ pull activé. PERMx à 0/pull désactivé.

Registre de choix de polarité du port M

$0255	PPSM7	PPSM6	PPSM5	PPSM4	PPSM3	PPSM2	PPSM1	PPSM0	**PPSM**
Reset	0	0	0	0	0	0	0	0	

Ce registre choisit entre pull-up et pull-down. PPSMx à 1/ pull-down. PPSMx à 0/ pull-up.

Registre de mode ou-câblé du port M

$0256	WOMM7	WOMM6	WOMM5	WOMM4	WOMM3	WOMM2	WOMM1	WOMM0	**WOMM**
Reset	0	0	0	0	0	0	0	0	

WOMMx à 1 / sortie x collecteur ouvert. à 0 / sortie normale.

Registre de redirection des modules

$0257	0	MODRR6	MODRR5	MODRR4	MODRR3	MODRR2	MODRR1	MODRR0	**MODRR**
Reset	0	0	0	0	0	0	0	0	

Permet de rediriger les registres de certains modules vers des adresses libres. Voir notice détaillée.

Registre d'entrée sortie du port P

$0258	PTP7	PTP6	PTP5	PTP4	PTP3	PTP2	PTP1	PTP0	**PTP**
Reset	0	0	0	0	0	0	0	0	
PWM	PWM7	PWM6	PWM5	PWM4	PWM3	PWM2	PWM1	PWM0	
SPI	SCK2	SS2	MOSI2	MISO2	SS1	SCK1	MOSI1	MISO1	

Si le bit de direction de donnée de la patte est à 1, une lecture donne la valeur du registre de port, sinon elle donne la valeur de la patte.

Registre d'entrée du port P

$0259	PTIP7	PTIP6	PTIP5	PTIP4	PTIP3	PTIP2	PTIP1	PTIP0	**PTIP**
Reset	-	-	-	-	-	-	-	-	

L'écriture dans ce registre est sans effet. Ce registre donne toujours l'état des pattes associées.

Registre de direction de données du port P

$025A	DDRP7	DDRP	DDRP	DDRP	DDRP	DDRP	DDRP	DDRP	**DDRP**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure la direction de chaque patte du port P. DDRPx à 1/ patte x en sortie. DDRPx à 0/ patte x en entrée

Registre de réduction de sortance du port P

$025B	RDRP7	RDRP6	RDRP5	RDRP4	RDRP3	RDRP2	RDRP1	RDRP0	**RDRP**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure la sortance de chaque patte en sortie . RDRPx à 1/ sortance réduite au tiers. RDRPx à 0 /sortance complète.

Registre de validation du pull du port P

$025C	PERP7	PERP6	PERP5	PERP4	PERP3	PERP2	PERP1	PERP0	**PERP**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure l'activation d'un pull-up ou d'un pull-down en entrée. PERPx à 1/ pull activé. PERPx à 0/pull désactivé.

Registre de choix de polarité du port P

$025D	PPSP7	PPSP6	PPSP5	PPSP4	PPSP3	PPSP2	PPSP1	PPSP0	**PPSP**
Reset	0	0	0	0	0	0	0	0	

Ce registre a deux usages : choisir entre pull-up et pull-down et choisir la polarité du front d'interruption actif.
PPSPx à 1/ pull-down et front montant. PPSPx à 0/ pull-up et front descendant

Registre de validation des interruptions du port P

$025E	PIEP7	PIEP6	PIEP5	PIEP4	PIEP3	PIEP2	PIEP1	PIEP0	**PIEP**
Reset	0	0	0	0	0	0	0	0	

Ce registre valide ou invalide pour chaque patte le front actif d'interruption externe.
PIEPx à 1/interruption validée. PIEPx à 0/interruption invalidée

Registre d'indicateurs d'interruptions du port P

$025F	PIFP7	PIFP6	PIFP5	PIFP4	PIFP3	PIFP2	PIFP1	PIFP0	**PIFP**
Reset	0	0	0	0	0	0	0	0	

Chaque indicateur est mis sur un front actif de la patte d'entrée associée (front montant ou descendant selon le registre PPSP).
Pour remettre le bit PIFPx à 0, écrire un "1" dessus. L'écriture d'un 0 est sans effet.

Registre d'entrée sortie du port H

$0260	PTH7	PTH6	PTH5	PTH4	PTH3	PTH2	PTH1	PTH0	**PTH**
Reset	0	0	0	0	0	0	0	0	

Si le bit de direction de donnée de la patte est à 1, une lecture donne la valeur du registre de port, sinon elle donne la valeur de la patte.

Registre d'entrée du port H

$0261	PTIH7	PTIH	PTIH	PTIH	PTIH	PTIH	PTIH	PTIH	**PTIH**
Reset	-	-	-	-	-	-	-	-	

L'écriture dans ce registre est sans effet. Ce registre donne toujours l'état des pattes associées.

Registre de direction de données du port H

$0262	DDRH7	DDRH	DDRH	DDRH	DDRH	DDRH	DDRH	DDRH	**DDRH**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure la direction de chaque patte du port H. DDRHx à 1/ patte x en sortie. DDRHx à 0/ patte x en entrée

Registre de réduction de sortance du port H

$0263	RDRH7	RDRH	RDRH	RDRH	RDRH	RDRH	RDRH	RDRH	**RDRH**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure la sortance de chaque patte en sortie . RDRHx à 1/ sortance réduite au tiers. RDRHx à 0 /sortance complète.

Registre de validation du pull du port H

$0264	PERH7	PERH6	PERH5	PERH4	PERH3	PERH2	PERH1	PERH0	**PERH**
Reset	0	0	0	0	0	0	0	0	

Ce registre configure l'activation d'un pull-up ou d'un pull-down en entrée. PERHx à 1/ pull activé. PERHx à 0/pull désactivé.

Registre de choix de polarité du port H

$0265	PPSH7	PPSH6	PPSH5	PPSH4	PPSH3	PPSH2	PPSH1	PPSH0	**PPSH**
Reset	0	0	0	0	0	0	0	0	

Ce registre a deux usages : choisir entre pull-up et pull-down et choisir la polarité du front d'interruption actif.
PPSHx à 1/ pull-down et front montant. PPSHx à 0/ pull-up et front descendant

Registre de validation d'interruptions du port H

$0266	PIEH7	PIEH6	PIEH5	PIEH4	PIEH3	PIEH2	PIEH1	PIEH0	**PIEH**
Reset	0	0	0	0	0	0	0	0	

Ce registre valide ou invalide pour chaque patte le front actif d'interruption externe.
PIEHx à 1/interruption validée. PIEHx à 0/interruption invalidée

Registre d'indicateurs d'interruptions du port H

$0267	PIFH7	PIFH6	PIFH5	PIFH4	PIFH3	PIFH2	PIFH1	PIFH0	**PIFH**
Reset	0	0	0	0	0	0	0	0	

Chaque indicateur est mis sur un front actif de la patte d'entrée associée (front montant ou descendant selon le registre PPSH).
Pour remettre le bit PIFHx à 0, écrire un "1" dessus. L'écriture d'un 0 est sans effet.

Registre d'entrée sortie du port J

$0268	PTJ7	PTJ6	0	0	0	0	PTJ1	PTJ0	**PTJ**
Reset	0	0	-	-	-	-	0	0	
CAN	TXCAN4	RXCAN4							
IIC	SCL	SDA							

Si le bit de direction de donnée de la patte est à 1, une lecture donne la valeur du registre de port, sinon elle donne la valeur de la patte.

Registre d'entrée du port J

$0269	PTIJ7	PTIJ6	0	0	0	0	PTIJ1	PTIJ0	**PTIJ**
Reset	-	-	-	-	-	-	-	-	

L'écriture dans ce registre est sans effet. Ce registre donne toujours l'état des pattes associées.

Registre de direction de données du port J

$026A	DDRJ7	DDRJ6	0	0	0	0	DDRJ1	DDRJ0	**DDRJ**
Reset	0	0	-	-	-	-	0	0	

Ce registre configure la direction de chaque patte du port J. DDRJx à 1/ patte x en sortie. DDRJx à 0/ patte x en entrée

Registre de réduction de sortance du port J

$026B	RDRJ7	RDRJ6	0	0	0	0	RDRJ1	RDRJ0	**RDRJ**
Reset	0	0	-	-	-	-	0	0	

Ce registre configure la sortance de chaque patte en sortie . RDRJx à 1/ sortance réduite au tiers. RDRJx à 0 /sortance complète.

Registre de validation du pull du port J

$026C	PERJ7	PERJ	0	0	0	0	PERJ	PERJ	**PERJ**
Reset	0	0	-	-	-	-	0	0	

Ce registre configure l'activation d'un pull-up ou d'un pull-down en entrée. PERJx à 1/ pull activé. PERJx à 0/pull désactivé.

Registre de choix de polarité du port J

$026D	PPSJ7	PPSJ6	0	0	0	0	PPSJ1	PPSJ0	**PPSJ**
Reset	0	0	-	-	-	-	0	0	

Ce registre a deux usages : choisir entre pull-up et pull-down et choisir la polarité du front d'interruption actif.
PPSJx à 1/ pull-down et front montant. PPSJx à 0/ pull-up et front descendant

Registre de validation d'interruptions du port J

$026E	PIEJ7	PIEJ6	0	0	0	0	PIEJ1	PIEJ0	**PIEJ**
Reset	0	0	-	-	-	-	0	0	

Ce registre valide ou invalide pour chaque patte le front actif d'interruption externe.
PIEJx à 1/interruption validée. PIEJx à 0/interruption invalidée

Registre d'indicateurs d'interruptions du port J

$026F	PIFJ7	PIFJ6	0	0	0	0	PIFJ1	PIFJ0	**PIFJ**
Reset	0	0	-	-	-	-	0	0	

Chaque indicateur est mis sur un front actif de la patte d'entrée associée (front montant ou descendant selon le registre PPSJ).
Pour remettre le bit PIFJx à 0, écrire un "1" dessus. L'écriture d'un 0 est sans effet.

Carte d'application EVB12

Les schémas de la carte de TP qui figurent à la suite sont tirés des fichiers

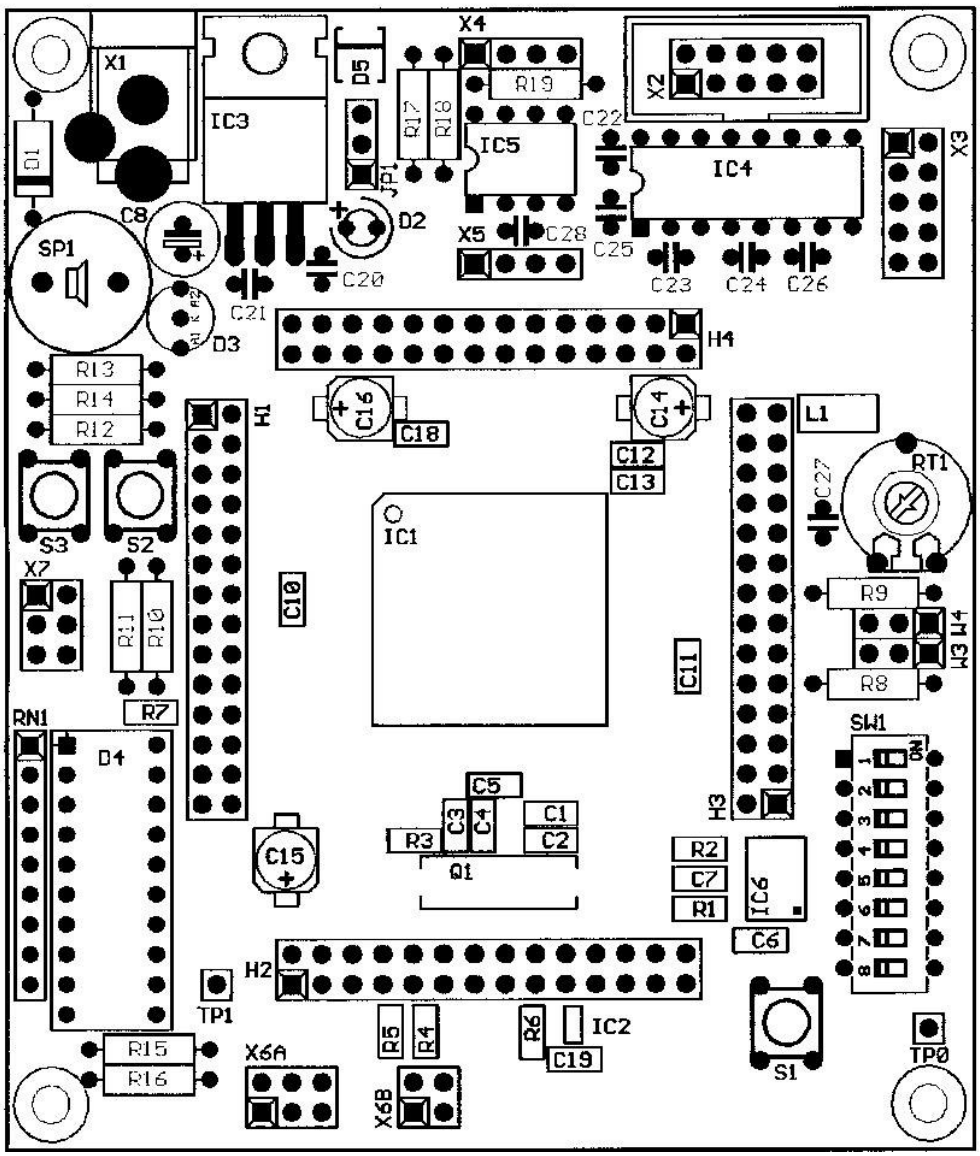
HCS12 TBV100_MANUAL.PDF

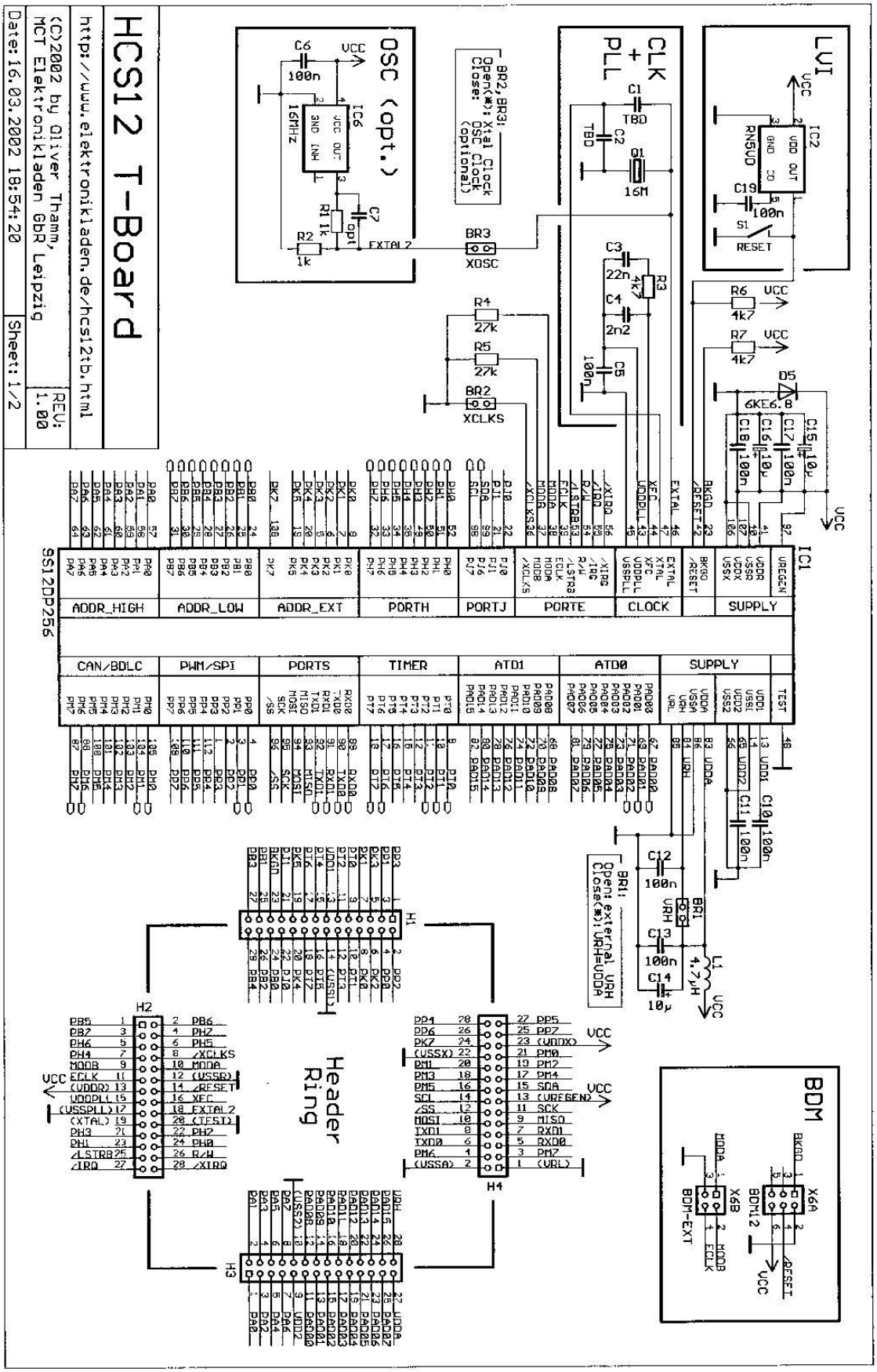
et

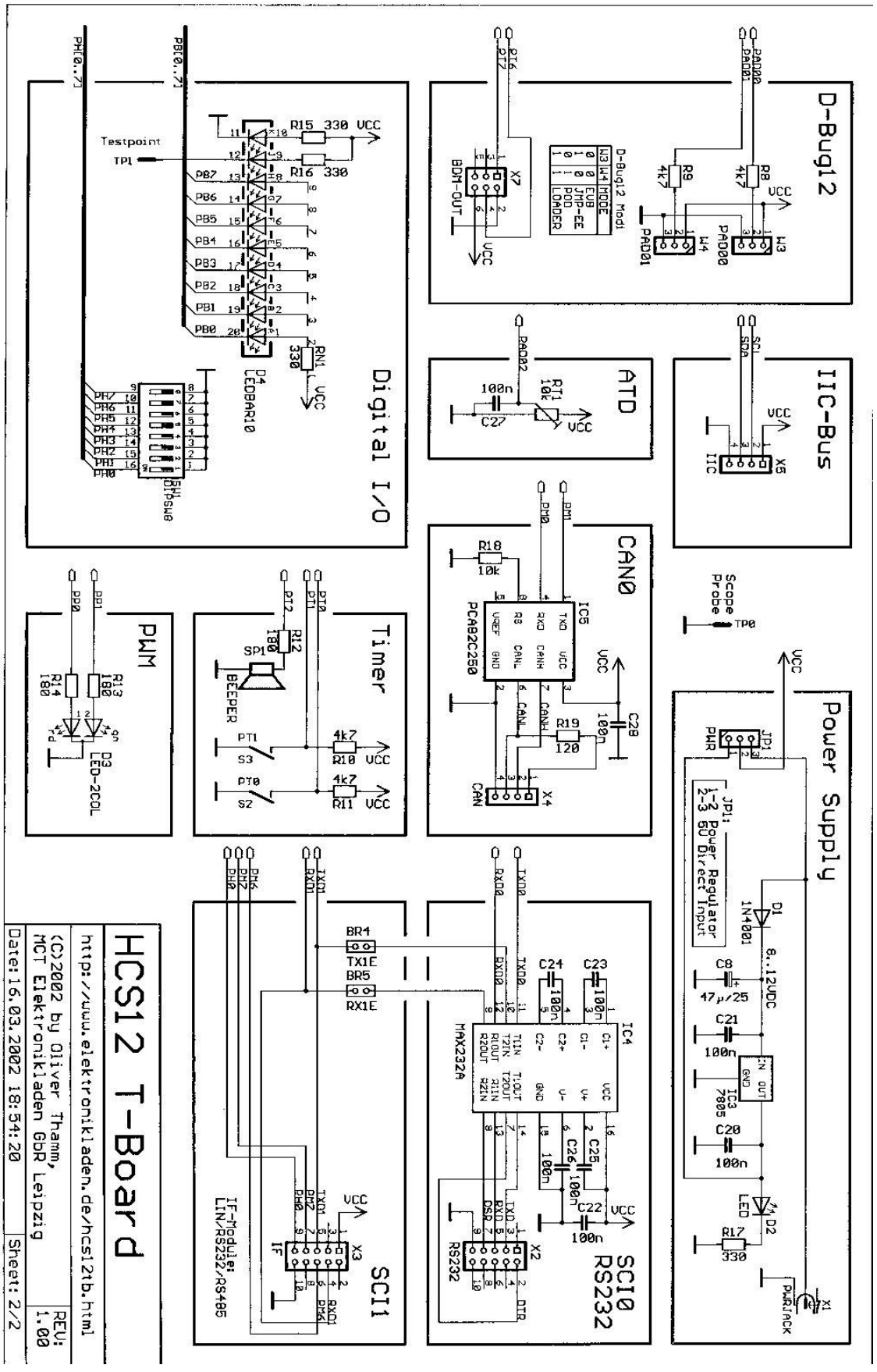
HCS12TVB100_SCHEMA.PDF

Ils décrivent la carte originale nue, sans préjuger des annexes qu'on sera amené à y ajouter durant les TP.

Implantation des composants




HCS12 T-Board
http://www.elektroniktaden.de/hcs12tb.html
(C)2002 by Oliver Thamm,
MCT Elektroniktaden GbR Leipzig
Date: 16.03.2002 18:54:20
REV: 1.00
Sheet: 1/2
LVI
CLK + PLL
OSC (opt.)
BDM
Header Ring
SS12DP256
ADDR_HIGH
ADDR_LOW
ADDR_EXT
PORTH
PORTJ
PORTE
CLOCK
SUPPLY
CAN/BDLC
PWM/SPI
PORTS
TIMER
ATD1
ATD0
SUPPLY
RESET
XOSC
XCLKS
EXTAL2
16MHz
BDM-EXT
BDM12


D-Bug12
IIC-Bus
ATD
Digital I/O
CAN0
Power Supply
Timer
PWM
SCI0 RS232
SCI1
HCS12 T-Board
http://www.elektronikladen.de/hcs12tb.html
(C)2002 by Oliver Thamm,
MCT Elektronikladen GbR Leipzig
Date:16.03.2002 18:54:20
REV: 1.00
Sheet: 2/2

Codage des instructions du HC12

Page 1 bas	0X	1X	2X	3X	4X	5X	6X	7X
0	BGND	ANDCC	BRA	PULX	NEGA	NEGB	NEG ind	NEG ext
1	MEM	EDIV	BRN	PULY	COMA	COMB	COM ind	COM ext
2	INY	MUL	BHI	PULA	INCA	INCB	INC ind	INC ext
3	DEY	EMUL	BLS	PULB	DECA	DECB	DEC ind	DEC ext
4	DBEQ-DBNE	ORCC	BCC-BHS	PSHX	LSRA	LSRB	LSR ind	LSR ext
5	JMP ind	JSR ind	BCS-BLO	PSHY	ROLA	ROLB	ROL ind	ROL ext
6	JMP ext	JSR ext	BNE	PSHA	RORA	RORB	ROR ind	ROR ext
7	BSR	JSR dir	BEQ	PSHB	ASRA	ASRB	ASR ind	ASR ext
8	INX	PO	BVC	PULC	ASLA-LSLA	ASLB-LSLB	ASL-LSL ind	ASL-LSL ext
9	DEX	LEAY	BVS	PSHC	LSRD	ASLD-LSLD	CLR ind	CLR ext
A	RTC	LEAX	BPL	PULD	CALL ext	STAA dir	STAA ind	STAA ext
B	RTI	LEAS	BMI	PSHD	CALL ind	STAB dir	STAB ind	STAB ext
C	BSET ind	BSET ext	BGE	wavi	BSET dir	STD dir	STD ind	STD ext
D	BCLR ind	BCLR ext	BLT	RTS	BCLR dir	STY dir	STY ind	STY ext
E	BRSET ind	BRSET ext	BGT	WAI	BRSET dir	STX dir	STX ind	STX ext
F	BRCLR ind	BRCLR ext	BLE	SWI	BRCLR dir	STS dir	STS ind	STS ext
Page1 haut	**8x**	**9x**	**Ax**	**Bx**	**Cx**	**Dx**	**Ex**	**Fx**
0	SUBA imm	SUBA dir	SUBA ind	SUBA ext	SUBB imm	SUBB dir	SUBB ind	SUBB ext
1	CMPA imm	CMPA dir	CMPA ind	CMPA ext	CMPB imm	CMPB dir	CMPB ind	CMPB ext
2	SBCA imm	SBCA dir	SBCA ind	SBCA ext	SBCB imm	SBCB dir	SBCB ind	SBCB ext
3	SUBD imm	SUBD dir	SUBD ind	SUBD ext	ADDD imm	ADDD dir	ADDD ind	ADDD ext
4	ANDA imm	ANDA dir	ANDA ind	ANDA ext	ANDB imm	ANDB dir	ANDB ind	ANDB ext
5	BITA imm	BITA dir	BITA ind	BITA ext	BITB imm	BITB dir	BITB ind	BITB ext
6	LDAA imm	LDAA dir	LDAA ind	LDAA ext	LDAB imm	LDAB dir	LDAB ind	LDAB ext
7	CLRA	TSTA	NOP	TFR-EXG	CLRB	TSTB	TST	TST
8	EORA imm	EORA dir	EORA ind	EORA ext	EORB imm	EORB dir	EORB ind	EORB ext
9	ADCA imm	ADCA dir	ADCA ind	ADCA ext	ADCB imm	ADCB dir	ADCB ind	ADCB ext
A	ORAA imm	ORAA dir	ORAA ind	ORAA ext	ORAB imm	ORAB dir	ORAB ind	ORAB ext
B	ADDA imm	ADDA dir	ADDA ind	ADDA ext	ADDB imm	ADDB dir	ADDB ind	ADDB ext
C	CPD imm	CPD dir	CPD ind	CPD ext	LDD imm	LDD dir	LDD ind	LDD ext
D	CPY imm	CPY dir	CPY ind	CPY ext	LDY imm	LDY dir	LDY ind	LDY ext
E	CPX imm	CPX dir	CPX ind	CPX ext	LDX imm	LDX dir	LDX ind	LDX ext
F	CPS imm	CPS dir	CPS ind	CPS ext	LDS imm	LDS dir	LDS ind	LDS ext

Codage des instructions du HC12

Page 2	0X	1X	2X	3X	4X	5X	6X	7X
0	MOVW imm-ind	IDIV	LBRA	TRAP	TRAP	TRAP	TRAP	TRAP
1	MOVW ext-ind	FDIV	LBRN	TRAP	TRAP	TRAP	TRAP	TRAP
2	MOVW ind-ind	EMACS	LBHI	TRAP	TRAP	TRAP	TRAP	TRAP
3	MOVW imm-ext	EMULS	LBLS	TRAP	TRAP	TRAP	TRAP	TRAP
4	MOVW ext-ext	EDIVS	LBCC-LBHS	TRAP	TRAP	TRAP	TRAP	TRAP
5	MOVW ind-ext	IDIVS	LBCS-LBLO	TRAP	TRAP	TRAP	TRAP	TRAP
6	ABA	SBA	LBNE	TRAP	TRAP	TRAP	TRAP	TRAP
7	DAA	CBA	LBEQ	TRAP	TRAP	TRAP	TRAP	TRAP
8	MOVB imm-ind	MAXA	LBVC	TRAP	TRAP	TRAP	TRAP	TRAP
9	MOVB ext-ind	MINA	LBVS	TRAP	TRAP	TRAP	TRAP	TRAP
A	MOVB ind-ind	EMAXD	LBPL	REV	TRAP	TRAP	TRAP	TRAP
B	MOVB imm-ext	EMIND	LBMI	REVW	TRAP	TRAP	TRAP	TRAP
C	MOVB ext-ext	MAXM	LBGE	WAV	TRAP	TRAP	TRAP	TRAP
D	MOVB ind-ext	MINM	LBLT	TBL	TRAP	TRAP	TRAP	TRAP
E	TAB	EMAXM	LBGT	STOP	TRAP	TRAP	TRAP	TRAP
F	TBA	EMINM	LBLE	ETBL	TRAP	TRAP	TRAP	TRAP

Codage de l'adressage indexé

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	0,X	-16,X	1,+X	1,X+	0,Y	-16,Y	1,+Y	1,Y+	0,SP	-16,SP	1,+SP	1,SP+	0,PC	-16,PC	n,X	n,SP
x1	1,X	-15,X	2,+X	2,X+	1,Y	-15,Y	2,+Y	2,Y+	1,SP	-15,SP	2,+SP	2,SP+	1,PC	-15,PC	-n,X	-n,SP
x2	2,X	-14,X	3,+X	3,X+	2,Y	-14,Y	3,+Y	3,Y+	2,SP	-14,SP	3,+SP	3,SP+	2,PC	-14,PC	n,X	n,SP
x3	3,X	-13,X	4,+X	4,X+	3,Y	-13,Y	4,+Y	4,Y+	3,SP	-13,SP	4,+SP	4,SP+	3,PC	-13,PC	[n,X]	[n,SP]
x4	4,X	-12,X	5,+X	5,X+	4,Y	-12,Y	5,+Y	5,Y+	4,SP	-12,SP	5,+SP	5,SP+	4,PC	-12,PC	A,X	A,SP
x5	5,X	-11,X	6,+X	6,X+	5,Y	-11,Y	6,+Y	6,Y+	5,SP	-11,SP	6,+SP	6,SP+	5,PC	-11,PC	B,X	B,SP
x6	6,X	-10,X	7,+X	7,X+	6,Y	-10,Y	7,+Y	7,Y+	6,SP	-10,SP	7,+SP	7,SP+	6,PC	-10,PC	D,X	D,SP
x7	7,X	-9,X	8,+X	8,X+	7,Y	-9,Y	8,+Y	8,Y+	7,SP	-9,SP	8,+SP	8,SP+	7,PC	-9,PC	[D,X]	[D,SP]
x8	8,X	-8,X	8,-X	8,X-	8,Y	-8,Y	8,-Y	8,Y-	8,SP	-8,SP	8,-SP	8,SP-,SP+	8,PC	-8,PC	n,Y	n,PC
x9	9,X	-7,X	7,-X	7,X-	9,Y	-7,Y	7,-Y	7,Y-	9,SP	-7,SP	7,-SP	7,SP-	9,PC	-7,PC	-n,Y	-n,PC
xA	10,X	-6,X	6,-X	6,X-	10,Y	-6,Y	6,-Y	6,Y-	10,SP	-6,SP	6,-SP	6,SP-	10,PC	-6,PC	n,Y	n,PC
xB	11,X	-5,X	5,-X	5,X-	11,Y	-5,Y	5,-Y	5,Y-	11,SP	-5,SP	5,-SP	5,SP-	11,PC	-5,PC	[n,Y]	[n,PC]
xC	12,X	-4,X	4,-X	4,X-	12,Y	-4,Y	4,-Y	4,Y-	12,SP	-4,SP	4,-SP	4,SP-	12,PC	-4,PC	A,Y	A,PC
xD	13,X	-3,X	3,-X	3,X-	13,Y	-3,Y	3,-Y	3,Y-	13,SP	-3,SP	3,-SP	3,SP-	13,PC	-3,PC	B,Y	B,PC
xE	14,X	-2,X	2,-X	2,X-	14,Y	-2,Y	2,-Y	2,Y-	14,SP	-2,SP	2,-SP	2,SP-	14,PC	-2,PC	D,Y	D,PC
xF	15,X	-1,X	1,-X	1,X-	15,Y	-1,Y	1,-Y	1,Y-	15,SP	-1,SP	1,-SP	1,SP-	15,PC	-1,PC	[D,Y]	[D,PC]